ENERGY EFFICIENT, SECURE AND NOISE ROBUST DEEP LEARNING FOR THE INTERNET OF THINGS

A Dissertation
Presented to
The Academic Faculty

By

Taesik Na

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

August 2018

ENERGY EFFICIENT, SECURE AND NOISE ROBUST DEEP LEARNING FOR THE INTERNET OF THINGS

Approved by:

Dr. Saibal Mukhopadhyay, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Tushar Krishna
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Doug Burger
Computer Architecture Group
Microsoft Research

Dr. Santosh Pande
School of Computer Science
Georgia Institute of Technology

Date Approved: July 17, 2018

To my family

ACKNOWLEDGEMENTS

I am fortunate to have had so many great mentors, friends, colleagues and family members throughout my days at Georgia Tech.

First, I would like to thank my advisor, Prof. Saibal Mukhopadhyay for giving me the freedom of research and unconditional support during my PhD. He has always been a great supporter, mentor and source of knowledge. He has an ability that turns my preliminary research ideas into better ones and shed light on a subject when I didn't have any idea. He waited for me, and gave encouragement when I had been having trouble dealing with unfamiliar topics. I have enjoyed working and talking with him, and will never forget what he has done for me.

I would also like to thank my thesis committee members, Prof. Sudhakar Yalamanchili, Prof. Tushar Krishna, Prof. Santosh Pande and Dr. Doug Burger for their time and invaluable comments that greatly improved the quality of this thesis.

This thesis cannot be done without the help from many former and current members of GREEN lab. I am grateful to work with and to know Dr. Minki Cho, Dr. Denny Lie, Dr. Sergio Carlo, Dr. Wen Yueh, Dr. Amit Trivedi, Dr. Boris Alexandrov, Dr. Zakir Ahmed, Dr. Duckhwan Kim, and Dr. Jaeha Kim, and Dr. Monodeep Kar. I cherish the fond memories with the future PhDs, Jong Hwan Ko, Faisal Amir, Arvind Singh, Yun Long, Burhan Mudassar, Edward Lee, Venkata Chaitanya Krishna Chekuri, Minah Lee, Nihar Dasari, Priyabrata Saha, Nikhil Chwala, Xueyuan She, and Muhammad Arslan Ali. I especially thank Jong Hwan Ko who has been a great friend and colleague. I have been motivated to achieve more by seeing his endless hours of hard work. I wish all the best to their future research and other endeavors.

I would like to thank the members of Apostles Korean Baptist Church. I cherish the days with Paster Youngman Shon, Youngwol Shon, Dongryul Shon, Dongwon Shon, Andy Kim, Jain Yoo, Yoobin Kim, and Soohyon Min as beautiful memories. I especially thank

Paster Youngman Shon and Youngwol Shon who have given me unconditional love and prayers. They have helped me and prayed for me when I was in desperate need of God's help.

I give my sincere thanks to my parents, who have always been my supporters in my life either distant or nearby, for their endless trust, love and prayers. I would also like to thank my parents-in-law for supporting my research with their love and encouragement. I could never find enough words to truly express my gratitude to my beloved wife, Jinyoung Ahn, and my two little boys, Yuchan Na and Yushin Na, for giving me unforgettable moments and love. I dedicate this thesis to my beloved family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The objective of this research is to design an energy efficient, secure and noise robust deep learning system for the Internet of Things (IoTs). The research particularly focuses on energy efficient training of deep learning, adversarial machine learning, and noise robust deep learning. To enable energy efficient training of deep learning, the research studies impact of a limited precision training of various types of neural networks like convolutional neural networks (CNNs) and recurrent neural networks (RNNs). For CNNs, the work proposes dynamic precision scaling algorithm, and precision flexible computing unit to accelerate CNNs training. For RNNs, the work studies impact of various hyper-parameters to enable low precision training of RNNs and proposes low precision computing unit with stochastic rounding. To enhance the security of deep learning, the research proposes cascade adversarial machine learning and additional regularization using a unified embedding for image classification and low level (pixel level) similarity learning. Noise robust and resolution-invariant image classification is also achieved by adding this low level similarity learning. Mixture of pre-processing experts model is proposed for noise robust object detection network without sacrificing accuracy for the clean images.

CHAPTER 1
INTRODUCTION

Deep learning [1] has been very successful in many scientific domains such as image classification [2, 3, 4, 5, 6, 7], image segmentation [8], image caption generation [9], machine translation [10, 11, 12], speech recognition [13, 14], and even games [15, 16]. Meanwhile, with a growing interest of deploying IoT devices and ever increasing data created from those devices, there is a need to shift deep learning computation from the cloud to the edge. There are several obstacles to overcome for the successful deep learning development and deployment for the internet of things.

First, fast and energy efficient training of deep learning is necessary to achieve realistic training time and to minimize power dissipation. Recent deep learning advances have been enabled by the enormous data set and gigantic deep learning models at the expense of increased training time. As deeper models tend to produce better performance, computational demand is getting more increased. Many prior works have proposed to use low precision arithmetic to accelerate the computation at the expense of the accuracy drop of a model. Majority of the works have studied the impact of low precision for convolutional neural networks (CNNs) [17] inference. Those works have leveraged quantization error tolerant characteristics of the neural networks. However, use of low precision arithmetic for the inference does not guarantee the feasibility of the low precision training. Thus, further study is necessary to speed up deep learning training and to improve the energy efficiency at the same time.

With a growing interest in sensor data monitoring, recurrent neural networks (RNNs) [18, 19] are heavily used as these networks are good for dealing with time series data. Unlike CNNs, RNNs require more computation to train than feed-forward networks due to the vanishing and exploding gradient problems [20]. To address this challenge, various

hardware approaches have been proposed for efficient computation of RNN models [21, 22]. Yet, many prior work use low precision only in the forward pass, requiring floating point precision for updating the parameters during backward propagation. Further study is also necessary for RNNs to enable entire low precision training for both forward pass and backward pass.

Second, security of the deep learning against potential adversarial attacks has to be enhanced. Recent works [23, 24, 25] have shown that todays deep neural networks are vulnerable to the inputs carefully generated by the adversaries. Most times, those inputs are not distinguishable to the human eyes, thus, it is possible that the system fails without producing noticeable changes. This reveals an astonishing difference in the information processing of humans and machines and raises security concerns for many deployed machine vision systems.

Injecting adversarial examples during training (adversarial training), [23, 24, 25] increases the robustness of a network against adversarial attacks. The networks trained with adversarial examples have shown noticeable robustness against the same type of adversarial attacks used in training. However it is shown to be very difficult to enhance robustness against unknown attacks at test time. Countermeasures for all types of adversarial attacks are necessary to design practical system of deep learning for the safe critical applications like autonomous driving vehicles.

Finally, noise robust and resolution-invariant computer vision is also an essential ingredient for successful deep learning deployment for the IoTs especially for the edge devices under stingy energy budget. For power hungry edge devices, it is critical to manage the trade-off between energy and quality of the captured image. Region of interest (RoI) based coding is becoming a norm for controlling the energy quality trade-off in resource constraint edge devices [26]. Also, inherent image sensor noise has to be considered for successful image classification for low-end devices [27]. Energy efficient and noise robust object detection network is necessary for object tracking on an edge device for round-the-

clock out door application.

The goal of this research is to design an energy-efficient, secure and noise robust deep learning for the IoTs. The research particularly focuses on energy efficient training of deep learning, adversarial machine learning, and noise robust deep learning. To increase the energy efficiency of the deep learning training, the research first proposes dynamic precision scaling algorithm for efficient training of convolutional neural networks (CNNs) and precision flexible multiplier-accumulator (MAC) unit. Limited precision training of recurrent neural networks (RNNs) is also studied. The work studies impact of various hyper-parameters to enable low precision training of RNNs and proposes low precision computing unit with stochastic rounding. The research also proposes cascade adversarial machine learning technique regularized with a unified embedding for image classification and low-level similarity learning to make deep learning robust against adversarial attacks. Noise robust and resolution-invariant image classification is realized by applying low-level similarity learning. The work proposes use of mixture of pre-processing experts model to enhance noise robustness for the object detection problem.

The rest of the thesis is organized as follows: In Chapter 2, the detailed background and literature survey are presented. Chapter 3 presents techniques to accelerate CNNs training. Chapter 4 presents the study of low precision training of RNNs. Chapter 5 introduces techniques to enhance robustness of machine learning against adversarial attacks. Chapter 6 presents noise-robust and resolution-invariant algorithms for image classification. Chapter 7 presents an algorithm to enhance noise robustness for an object detection network without sacrificing detection accuracy for normal clean images. Finally, Chapter 8 describes the key research contributions and provides future research direction.

CHAPTER 2
BACKGROUND

2.1 Energy Efficient Deep Learning

Deep neural networks [1] can be divided into two categories, feed-forward neural networks wherein connections between the units do not form a cycle and recurrent neural networks (RNNs) where connections between units form a directed graph. Among feed-forward neural networks, this thesis focuses on convolutional neural networks (CNNs) [17] since majority of breakthroughs in many machine learning tasks were enabled by CNNs. For recurrent neural networks, the study doesn't consider vanilla RNN since training vanilla RNN is known to be difficult due to gradient vanishing problem [20]. Rather, the research considers long-short-term memory (LSTM) [18] and gated recurrent unit (GRU) [19] as target RNNs since their advanced architecture eased difficulty of training, thus, now those are gaining popularity.

There have been many works to improve energy efficiency of deep learning. Prior works to improve energy efficiency can be divided into four categories, algorithms for efficient inference, hardware for efficient inference, algorithms for efficient training and hardware for efficient training.

2.1.1 Algorithms for Efficient Inference

Feed-Forward Neural Networks

As the networks becomes deeper and deeper [4], memory bandwidth and convolution computation becomes bottleneck in CNN, thus, many prior works have been proposed to optimize convolution operation and/or memory requirements.

[28] proposed the method to squeeze the weights of the neural networks for efficient

inference. In their work, they first pruned unimportant connections during training, quantized weights, applied weight sharing and applied Huffman encoding. With these methods, the size of the weights can be reduced by 30x 50x without losing accuracy.

[29] proposed to compress the weights with image compression method observing the weights of the fully connected layers can be seen as images. In their work, they used adaptive quality factors (high quality factors for important connections and low-quality factors for unimportant connections with gradients information) when applying image compression method and showed 40x compression ratio with small accuracy loss.

[30] proposed accelerating CNN by computing convolution operation in frequency domain since computationally heavy convolution can be changed simple elementwise multiplication in frequency domain. [31] even proposed entire frequency domain computation for CNN which further improves energy efficiency. Recent work exploited quantization even with one or two bits enabling further energy efficiency [32, 33, 34]. In some test cases, binarized network showed comparable results with floating point counterpart since binarized network acts as regularizer.

Recurrent Neural Networks

A few works have been done for energy efficient inference for RNNs compared to CNNs. Among them, [22] have shown that limited numerical precision can be applied in RNNs to reduce computation. To this end, the authors quantized weights and eliminated multiplications in the forward pass. This work applied low precision only in the forward pass, requiring floating point precision for backward propagation

2.1.2 Hardware for Efficient Inference

Feed-Forward Neural Networks

There have been many works utilizing dedicated hardware for efficient CNN inference. Most of the works exploited reusing the weights to minimize outside memory access and

used reduced precision to accommodate as many computation units as possible given area.

[35, 36] proposed to use dedicated hardware accelerators using to improve energy efficiency for CNN inference. [35] utilized roofline model, identified all possible solutions in the design space, and showed increase in throughput of the CNN. [36] proposed to use processing engines in a logical 3D rather than 2D topology. The drawback of this scheme is the routing complexity, but, the authors addressed this issue by reusing data to minimize routing complexity.

Custom accelerators have been also proposed to further improve the energy efficiency [37, 38, 39, 40]. [37] used compressed networks as target networks and applied sparse and weight sharing to reduce memory access by fitting weights in on-chip SRAM. [38] used multi-chip, embedded DRAM (EDRAM) and router fabric to fit larger models. [39] utilized row stationary dataflow to reduce memory access showing 1.4x - 2.5x lower energy than any other dataflows. [40] proposed use of accelerators for both convolutional layer and fully connected layer and RISC controllers for CNN inference. Authors in [40] utilized dynamic fixed-point format to efficiently represent weights for different statistics.

Recurrent Neural Networks

For RNNs, [41] used FPGA to accelerate LSTM inference. The authors optimized computation and communication demands for LSTM, and proposed a hardware architecture for efficient computation and data movement. [42] proposed the way to accelerate compressed LSTM inference on FPGA. First the authors used load balance aware pruning and dynamic precision data quantization. And they proposed a hardware architecture that can work directly on the sparse model to efficiently handle irregular computation pattern after compression.

[21] have introduced a programmable RNN using resistive random access memory (ReRAM). It showed speed-up of training with less energy by using this new type of memory. However, authors showed that the device variation of ReRAM can significantly de-

grade the system performance.

2.1.3 Algorithms for Efficient Training

Deeper models with larger parameters have shown to achieve better accuracy. However, the drawback of using deeper model is longer training time. For efficient training, [43] introduced a way, referred as batch normalization, to accelerate training of deep learning by reducing internal covariate shift. With batch normalization, the mean and variance of the internal nodes can be modified to be favorable for training. This allows us to have higher learning rate which enables faster training. Batch normalization also regularizes the model, thus, reduces need for using other regularization methods like Dropout [44].

[45] proposed a way of training networks, referred as Dense-Sparse-Dense (DSD), in three steps. It trains networks first with dense parameters, second with sparse parameters after pruning, and again with dense parameters. Authors showed better accuracy for the network trained with DSD than that with standard training. The intuition of this work is to learn important parameters first, and learn the rest of the parameters with already learned important parameters.

Another approach for efficient training is to use low precision arithmetic which is less computationally expensive than floating point counterpart. [46] proposed use of limited precision computation with fixed-point arithmetic and stochastic rounding. [47] has shown that dynamic-fixed-point arithmetic can help lower precision during training. Given a network and dataset, authors in [46, 47] have performed exhaustive training experiments by changing precision (total bit-width) and reported minimum usable precision.

2.1.4 Hardware for Efficient Training

Many hardware assisted solutions have been proposed to accelerate training such as distributed computing [48, 49] and GPU accelerated computing [2].

2.2 Secure Deep Learning

Deep neural networks and other machine learning classifiers are shown to be vulnerable to small perturbations to inputs [50, 51, 23, 52, 24]. Those inputs can be indistinguishable to the human eye, thus, it is possible that the system fails without producing any noticeable changes.

The adversarial examples can be generated by perturbing the inputs in one step or iteratively to either minimize confidence on true labels or increase confidence of a target false label. Those inputs intended to fool the model M1, are often misclassified on another model M2. This transferability of the adversarial examples means attacks for the target network without accessing its exact model can be easily possible with an adversarial examples generated from another networks. [52, 53] showed this black box attack in a realistic environment.

Previous works [23, 24, 54] have shown that injecting adversarial examples during training (adversarial training) increases the robustness of a network against adversarial attacks. The networks trained with ”one-step” method have shown noticeable robustness against ”one-step” attacks, but, limited robustness against ”iterative” attacks at test time. Another approach to increase robustness is to use defensive distillation to train network [55]. Improving robustness of the machine learning against various types of adversarial attacks remains still challenging and further research is necessary.

2.3 Noise-Robust and Resolution-Invariant Image Classification and Object Detection

Image classification using deep learning is being widely adopted for the internet of things (IoTs) [31]. For power hungry edge devices, managing the trade off between power consumption and the quality of the captured images is of great concern. Unprecedented controllability of the camera parameters [56] enables smart control of sensor data generation to

optimize the information given power budget. For example, region of interest (RoI) based coding at the sensor [26] can be enabled by controlling spatial resolution of the camera achieving loss of information with minimized energy consumption. Also, inherent image sensor noise has to be considered for successful image classification for low-end devices [27].

Many prior work have been studied the impact of low quality images on the image classification. Traditional denoising techniques like GSM [57] , KSVD [58] and BM3D [59] can be used to enhance the quality of the images as a pre-processing. Denoising auto-encoders [60, 61] have also been proposed. Super resolution [62] can be applied for low resolution images. Unnecessary process for the clean images due to the use of pre-processing for every input results in degraded accuracy for the clean images.

Data augmentation approaches [63, 64, 65] can also be used for both noisy images and low resolution images. Ratio of clean and perturbed images and loss formulation during training determines performance for each data distribution. The trade off between accuracy of the clean images and that of the perturbed images has to be considered.

Further study for the noise-robust object detection network is also necessary as there are not much works related with object detection for noisy and low resolution images.

CHAPTER 3
DYNAMIC PRECISION TRAINING OF CONVOLUTIONAL NEURAL NETWORKS

3.1 Introduction

Convolutional neural networks (CNNs) are attractive for many machine learning tasks like image classification, scene recognition, and natural language processing. The inference accuracy in CNN improves by adopting deeper network and more data; but at the expense of longer training time. Many hardware assisted solutions have been proposed to accelerate training such as distributed computing [48, 49] and GPU accelerated computing [2]. Dedicated hardware accelerator using FPGA [35] has also been proposed to improve computing throughput. Cong et al. proposed minimizing computation for CNN [66]. Lin et al. introduced CNN with fewer multiplications [67].

More recently, there is a growing interest in training deep neural network with limited precision. Gupta et al. proposed limited precision computation with fixed-point arithmetic and stochastic rounding [46]. Courbariaux et al. has shown that dynamic-fixed-point arithmetic can help lower precision during training [47].

Given a network and dataset, authors have performed exhaustive training experiments by changing total bit-width and reported minimum usable precision. However, to enable low-precision training, it is critical to develop an automated precision search algorithm instead of an exhaustive precision search as in [46] or [47], and an architecture that can improve system performance when operating at a lower precision.

This chapter explores hardware-assisted acceleration for training deep neural network by utilizing low-precision learning. A coupled algorithm and hardware approach that dynamically scales precision during training is introduced. The proposed approach utilizes

a precision-flexible multiply-and-accumulator (MAC) unit for computation. While low-precision learning has been explored before, the key contributions are:

1. Dynamic precision scaling (DPS) algorithm for training CNN is proposed. The proposed approach utilizes dynamic fixed point and successfully tracks good enough precision during training for an arbitrary network without exhaustive precision search.

2. A configurable MAC unit that can be configured for various precision (bit width) computation modes is proposed. The proposed hardware can enable faster computation by lowering latency for lower precision. A system architecture is presented for accelerated training using the proposed DPS algorithm and flexible MAC.

Design and synthesis results for the flexible MAC in 28nm CMOS are presented. The hardware power and performance results are evaluated after the post place and route. Simulation results show that 3.6x and 5.7x speed-up are achieved using the proposed algorithm and hardware in benchmark CNNs including Lenet [68] for handwritten digit classification on the MNIST dataset and Alexnet [2] for image style recognition on the Flickr images, respectively.

3.2 Proposed Approach: Concept

A fixed-point number is composed of integer part and fractional part and each part has its own bit width. The bit width for integer part is defined as IL and the bit-width for fractional part as FL. Fixed point format now can be represented as $< \texttt{IL, FL} >$ and the precision (total bit width) becomes IL+FL. The smallest positive number ϵ and the range for $< \texttt{IL, FL} >$ would be 2-FL and [-2IL-1, 2IL-1 - 2-FL], respectively.

The dynamic fixed-point format [69] allows several decimal points instead of single global one. With dynamic fixed-point format, the bit utilization can be optimized by minimizing IL enough not to overflow. For example, allocating longer bit width may not be needed for integer part to represent parameter values if most of them are below 1. As the

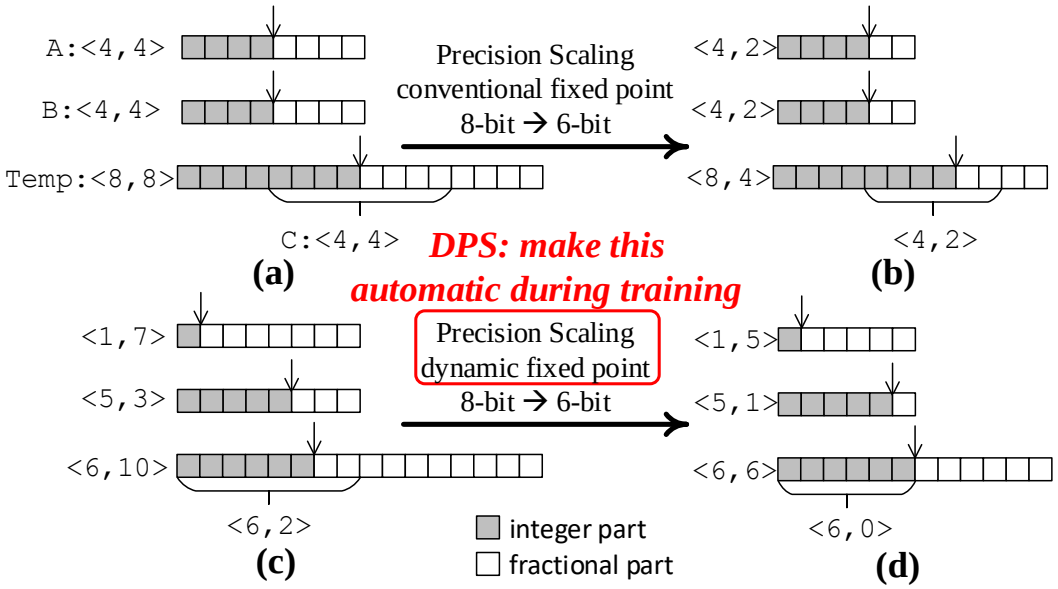


Figure 3.1: Example bit usage of (a) 8-bit, (b) 6-bit conventional fixed point, (c) 8-bit, and (d) 6-bit dynamic fixed point. (b) and (d) can be seen as precision (total bit width) scaling (8-bit $\rightarrow$ 6-bit) of (a) and (c) respectively.

parameter statistics in each layer of a CNN are different [70], it is beneficial for each layer to have its own optimized tuples $<$ IL, FL$>$ for parameters and data.

Figure 3.1 shows an example of precision scaling with conventional fixed point and dynamic fixed point when used in multiplication (A x B = C). As shown in this figure, dynamic fixed-point format allows different decimal point for each operand (A, B) and result (C), thus, can more efficiently utilize bit space. The proposed approach is motivated by the prior works that showed the feasibility of learning with low precision (i.e. reduce IL+FL) as well as utilization of dynamic fixed-point [7, 8]. During training of an arbitrary network, the proposed approach dynamically scales the precision (total bit-width) and allows dynamic fixed-point operation. The proposed approach of dynamically changing the precision of dynamic fixed point numbers will be referred to as the dynamic precision scaling (DPS). The minimized bit width by DPS, then, benefits from the proposed precision-flexible hardware that reduces computing time for lower precision.

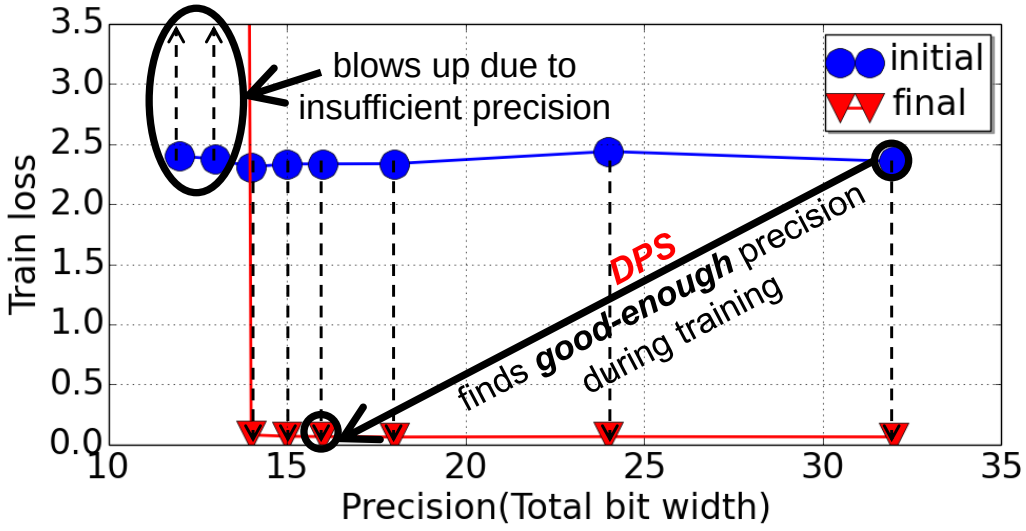


Figure 3.2: Train loss for Lenet [68] on MNIST dataset with dynamic fixed point. Blue line represents initial loss and the red line represents loss after training has done. Each dotted arrow shows single training trial without changing precision.

The objective is not to find minimum precision for a specific network by exhaustive precision search, but to speed up training with good enough precision. As shown in Figure 3.2, it is assumed that the minimum precision for a given network and dataset is not known, and start training with the maximum precision. Then, DPS algorithm tries to find good enough low precision to speed up training. A precision-flexible hardware platform is also proposed to achieve this goal. The research focuses on larger network size as capacity of the model increases with bigger networks which are in need of acceleration of training. For example, (number of parameters, number of MACs/forward pass) for well-known neural networks are: Lenet [68]- (431K, 2.3M), Alexnet [2]- (60M, 721M), and GoogLenet [4]- (6.8M, 1.5G)).

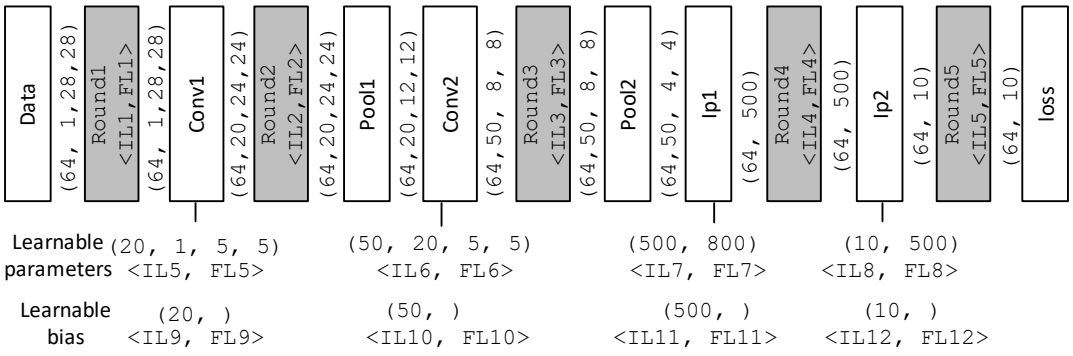


Figure 3.3: Fixed-point computation emulation with Caffe framework. Each round layer performs round-to-nearest operation with data blobs and diff blobs for forward pass and backward pass respectively.

3.3 Dynamic Precision Scaling

3.3.1 Fixed Point Arithmetic Emulation Setup

In this research, an existing deep learning software framework, Caffe [71] is used to emulate limited precision arithmetic. As data and derivatives flow through the network in the forward and backward passes, Caffe stores the information as blobs. A blob stores two chunk of memories, data and diff. The former is the data that is passed along, and the latter is the gradient. Round operation for both data and diff memories is performed to emulate fixed point arithmetic in a floating point based framework.

Figure 3.3 shows an example fixed-point emulation framework for Lenet. 4- or 2-length list on the forward/backward pass shows the size of dimensions for each blob. The size of the first dimension (in this example 64) in the network represents mini batch size for stochastic gradient decent (SGD) algorithm. And the rest shows size of each dimension of blob per single batch. 4-length list for learnable parameters in convolutional layers (Conv1, Conv2) represents number of filters (first dimension) and filter size (24 dimensions). 2-length list for learnable parameters in inner product layers (Ip1, Ip2) represents output and input size on forward/backward pass per single batch. Each learnable layer has 1

dimensional bias term and the value represents output size on forward/backward pass per single batch.

As shown in this figure, data (data), convolution (Conv1, Co nv2) and inner product (Ip1, Ip2) layers are followed by round layers (Round1-5). Round operation is only applied after all computation is done since the proposed MAC temporarily generates 2x larger precision output. Each round layer performs round-to-nearest operation based on $<$ `IL, FL`$>$ for data blobs and diff blobs in forward and backward pass respectively. Max pooling is used in the pooling layer, thus, it doesnt need following round layer.

Gradient terms (diff blobs in learnable layers) with respect to parameters are rounded before parameter updates to emulate fixed point arithmetic in the backward pass. And learnable parameters (data blobs in learnable layers) are rounded after the SGD updates.

3.3.2 Dynamic Precision Scaling Algorithm

This section describes how $<$ `IL, FL`$>$ for round layers and learnable parameters are scaled during the training. Overall, the algorithm controls dynamic fixed point format $<$ `IL, FL`$>$ for each layer and global precision during training. That means decimal point is controlled per layer and total bit width globally. The algorithm is activated only after forward/backward computation has done. The overhead of running this algorithm is very small since most of the training time is consumed in forward/backward computations in convolutional layers [72].

DPS algorithm first tries to find minimum ILs for all data paths and learnable parameters. Then, it aggressively scales precision to a given target length (tl: initial precision guess) which applies globally to all blobs in the net. If the moving average keeps decreasing, no action is taken. If the training becomes numerically unstable, it increases the precision to its maximum value (ml). Since the training might become extremely unstable due to the wrong guess (tl), immediate change to the maximum precision is beneficial. Once the training loss returns to its last minimum value through the maximum precision

training, DPS lowers the precision to the unit bit step (s) higher than target length (tl) not to exceed $ml - s$. In this way, the precision will eventually oscillate between ml and $ml - s$ alarming sufficient training has been done. From this point, it is up to the users to further fine-tune with a higher precision. ml, tl and s should be chosen based on the hardwares supportability or full benefit from the hardware cannot be achieved.

Algorithm 1 Dynamic Precision Scaling Algorithm

Require: P: List of pointers for round layers and learnable parameters, $window$: Moving average window for loss, tl: Target bit width, ml: Maximum bit width, s: Unit bit step.

```
initialize(ml)
mode ← IL_SEARCH
last_min_loss ← 0
repeat
    overFlowCheckRoundLayers()
    checkLearnableParams()
    ma_loss.push(computeMaLoss(window, loss))
    switch mode do
        case IL_SEARCH
            first_overflow ← 0
            for j in all round layers and learnable parameters
                first_overflow += ilSearch(P[j])
            if first_overflow == size(P) then
                mode ← FL_SEARCH
                precisionSet(tl)
        case FL_SEARCH
            if ma_loss[0] == max(ma_loss[0:window − 1]) then
                if current_precision < ml then
                    last_min_loss ← min(ma_loss[0:window − 1])
                    precisionSet(ml)
            else
                if ma_loss[0] < last_min_loss then
                    last_min_loss ← 0
                    if tl < ml − s then
                        tl ← tl + s
                    precisionSet(tl)
until training converged
```

Details of this heuristic are shown in Algorithm 1. The first step, initialize, initializes `IL` and `FL` with half of the maximum total bit width (ml) for each layer and learnable

parameter (line 1). ml should be large enough to ensure that the loss does not blow up and overflow does not occur at an initial stage. The state variables are also initialized ($mode, last\_min\_loss$) to their initial status (IL_SEARCH, 0) (line 23).

At every iteration during training regardless of mode, `overFlowCheckRoundLayers` and `checkLearnableParams` check overflows for all round layers and learnable parameters and adjust $<$ `IL, FL`$>$ s. The function calling order is important since the allowable minimum values of `IL`s in learnable parameters are limited by the `IL`s in the round layers. For example, if the `IL`s of two consecutive round layers are 3 and 4, `IL` of the learnable parameter between those two should be higher or equal to 1. The following function, `computeMaLoss`, calculates moving average for loss given window. This value is appended in ma_loss which will be used later.

Next, actions are determined based on the state variable mode (line 8). If the current mode is initial status (IL_SEARCH), `IL` is decreased by 1 for each round layer and parameter at every iteration up to the point where all round layers and learnable parameters have experienced their first overflows (line 1012). If all round layers and learnable parameters have found their optimal (minimum) `IL`s, the $mode$ is changed to FL_SEARCH. And total bit width is set to the target total bit width (tl) for all layers and learnable parameters by reducing `FL`s.

When the mode is FL_SEARCH, the global precision is scaled based on the training loss ($loss$). As training loss is very noisy, moving average loss (ma_loss) is utilized instead of raw loss value. If the ma_loss keeps decreasing, no action is taken. When the current moving average loss (ma_loss[0]) reaches its maximum value in given $window$ range (line 17), the precision is immediately changed to its maximum value (ml). By checking if ma_loss reaches its maximum given $window$, small perturbations in ma_loss are ignored. Once the current moving average loss reaches the last minimum moving average loss ($last\_min\_loss$) (line 22), the precision is changed again. But this time, the precision is scaled to the unit bit step (s) higher than target length (tl) not to exceed $ml - s$.

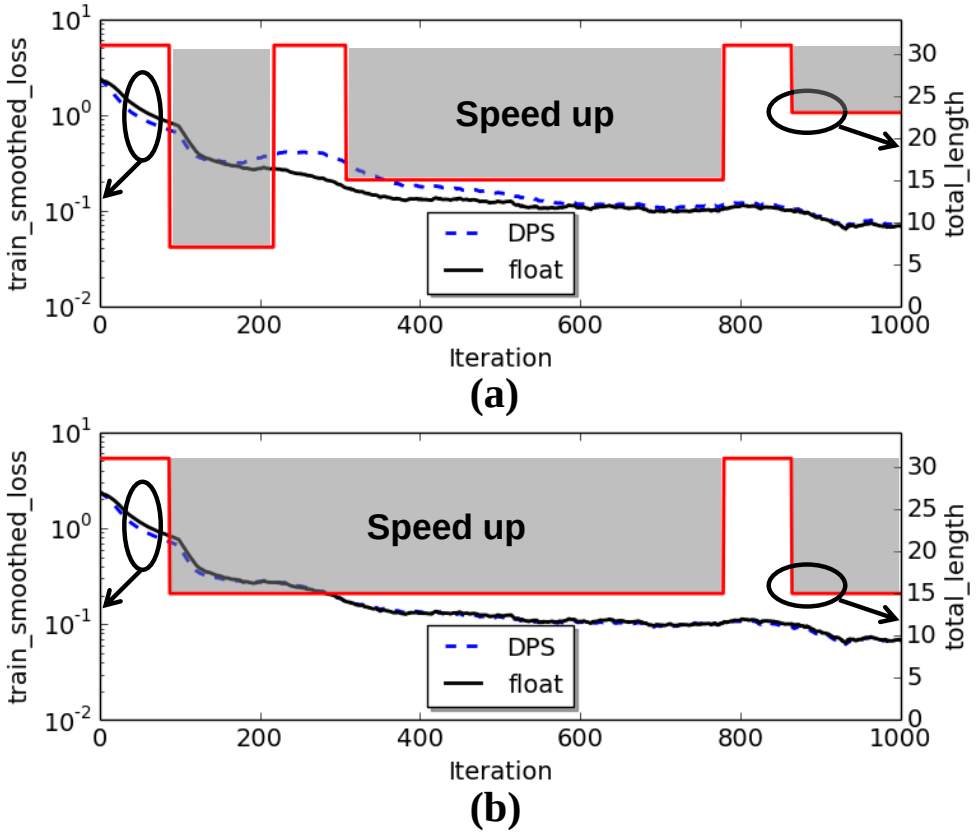


Figure 3.4: Moving average loss on the left y-axis and total bit-width w/o sign bit on the right y-axis on MNIST with DPS for (a) (tl:8, ml:32, s:8) and (b) (tl:16, ml:32, s:16).

The precision will eventually oscillate between ml and $ml - s$.

3.3.3 Dynamic Precision Scaling Results

DPS algorithm is applied to Lenet for MNIST hand-written digit classification and Alexnet for Flickr style recognition. Output layer size of Alexnet is modified from 1000 to 20 since Flickr style has 20 labels. Fine-tuning is performed using the weights learned from ImageNet 2012 dataset for Flickr style recognition.

MNIST

This section shows training results with MNIST dataset. SGD algorithm is used for the parameter updates and the batch size is set to 64. Figure 3.4 shows moving average loss

and the precision (total bit width) vs iteration plot. Figure 3.4 (a) represents DPS with (tl:8, ml:32, s:8) and Figure 3.4 (b) shows DPS with (tl:16, ml:32, s:16). For both cases, moving average window is set to 100. Baseline (training with floating point) result is also shown for comparison. The proposed DPS algorithm enables low precision training (shaded regions in the figure) for most of the training time. Shaded regions are the possible durations for speeding up the training when used with the proposed precision-flexible MAC which will be discussed later. The precision oscillates at the end of the training with the DPS algorithm. At that point, the users might consider whether to stop training or continue to train with high precision.

Training using fixed point format without DPS has also been performed as a comparison. $<$ `8, 8`$>$, $<$ `12, 12`$>$ and $<$ `16, 16`$>$ format are used for 16-bit, 24-bit and 32-bit fixed point respectively. Learning with 16-bit fixed point does not converge. Learning with 24-bit fixed point converges, but does not achieve maximum accuracy. Only 32-bit mode operation shows comparable results with floating point for MNIST data set.

Use of dynamic fixed point enables low precision training (16-bit) compared to conventional fixed point (32-bit) case for Lenet on MNIST. The role of DPS is to help find good enough precision with single training trial. This is very useful when the cost (time spent) on single training is very huge.

Flickr Style Recognition

In this section, simulation results on Flickr style images are shown. For this task, fine-tuning is performed. Training is initialized with the weights based on pre-trained weights from Alexnet on Imagenet database. 1,600 images are used for training and 400 images for the test.

Figure 3.5 shows moving training loss and the precision (total bit width) vs iteration plot. Figure 3.5(a) represents DPS with (tl:16, ml:64, s:16) and Figure 3.5(b) shows DPS with (tl:32, ml:64, s:32). For both cases, moving average window is set to 150. Baseline

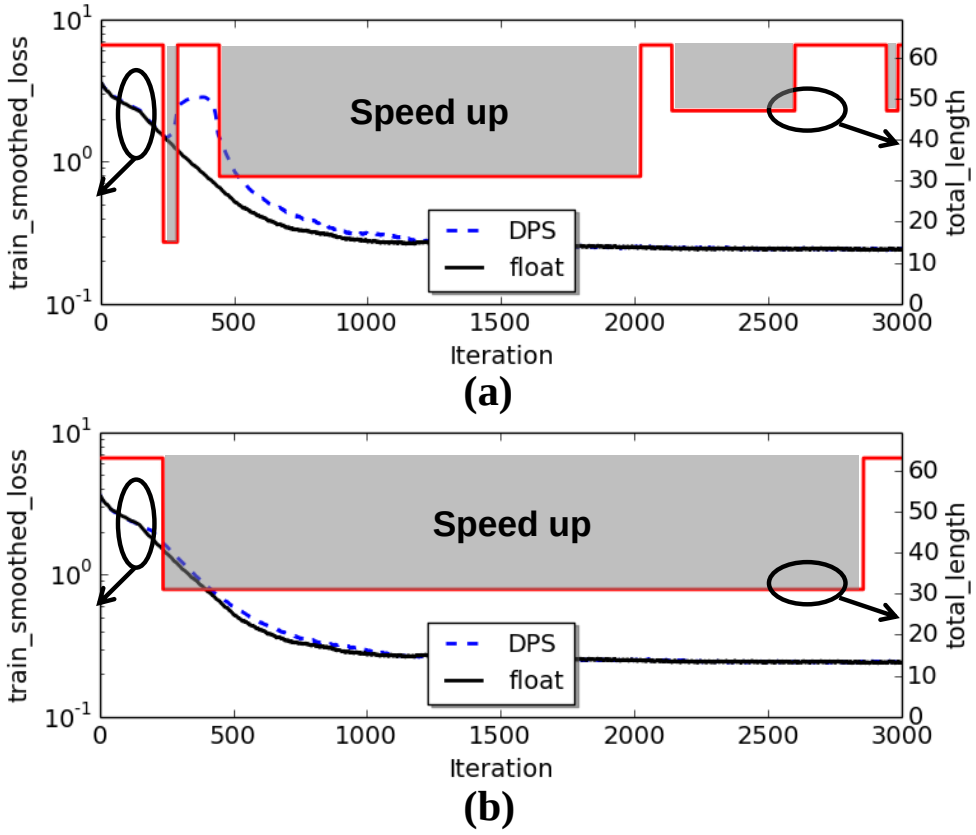


Figure 3.5: Moving average loss on left y-axis and total bit-width w/o sign bit on the right y-axis on Flickr style images with DPS for (a) (tl:16, ml:64, s:16) and (b) (tl:32, ml:64, s:32).

(training with floating point) result is also shown for comparison. This case also proves that the DPS algorithm successfully tracks good enough precision resulting in possible training speed-up with the proposed flexible MAC.

Training using fixed point format without DPS has also been performed as a comparison. $<$ `16, 16`$>$ and $<$ `32, 32`$>$ format are used for 32-bit and 64-bit fixed point respectively. Learning with 32-bit fixed point does not converge. Only 64-bit mode operation shows comparable results with floating point for Flickr style images.

Dynamic fixed point enables low precision training (32-bit) compared to conventional fixed point (64-bit) for modified Alexnet on Flickr style images. Compared with Lenet example, required precision increases as the network size increases or number of computation

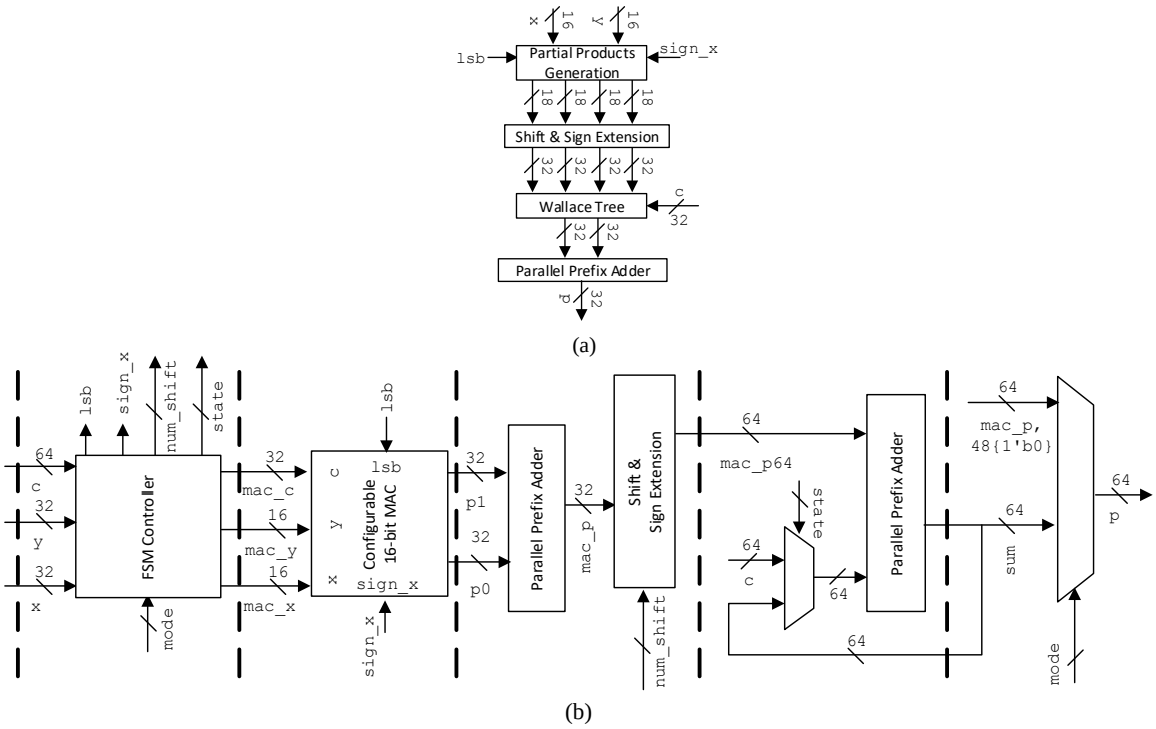


Figure 3.6: (a) 16-bit Configurable MAC (b) Pipelined 32/16-bit flexible MAC block diagram. Dotted lines represent registers. This MAC unit can perform 16-bit and 32-bit MAC operation based on $mode$ input.

per forward pass increases.

3.4 Flexible MAC Unit

3.4.1 Configurable MAC unit

Figure 3.6(a) shows a configurable radix-4 Booth-Wallace 16-bit MAC unit. Output `p` is calculated with an expression, `x*y+c`. 2 additional input ports `sign_x` and `lsb` are used which differs with the conventional MAC. `sign_x` tells how to treat `x` for partial product generation. If `sign_x` is 0, `x` is treated as an unsigned number and vice-versa. Thus, zero or `msb` of `x` is padded in front of `x` if `sign_x` is 0 or 1 respectively for partial product generation. `lsb` is used for the first radix-4 booth encoding. Normally zero is padded at the end of `y` and used for the first encoding, but, `lsb` is used instead of padding zero.

3.4.2 Flexible MAC

Figure 3.6(b) shows the proposed pipelined flexible 32-bit MAC with configurable 16-bit MAC unit as an example. It will be referred to as 32/16-bit flexible MAC for brevity. FSM controller determines all inputs for configurable 16-bit MAC unit based on `mode` (16-bit and 32-bit mode). For proper shift operation, it also generates `num_shift` which determines how much the partial product should be shifted. Since the critical path exists in the configurable 16-bit MAC unit, re-timing is performed on the 16-bit MAC. As a result, parallel prefix adder is placed at the next stage of the 16-bit MAC.

Latency: For conventional MAC, the number of cycles required to get the output is 1. For the proposed MAC, the number of cycles required for all partial products generation is 1 and 4 cycles for 16-bit and 32 bit operation respectively and the added latency is 2 and 4 for 16-bit and 32 bit mode due to the pipeline. For 32-bit computation mode, the additional cycles are required in (1) FSM controller unit, (2) shift & sign extension unit, (3) the 64-bit accumulator unit, and (4) the final mux stage. For 16-bit computation mode, shift & sign extension unit and 64-bit accumulator can be bypassed as the output can be just calculated in configurable 16-bit MAC unit. Resulting Latencies of proposed flexible MAC are 1+2 and 4+4 cycles for 16-bit and 32 bit mode.

Throughput: throughput for each mode will be the clock frequency divided by the latency for general purpose MAC unit. For the proposed precision flexible MAC, effective throughput is reduced due to the additional latency for each computation mode. However, the additional latency can be hided as it is pipelined and the partial products should be accumulated. Generally, the number of partial products summation is large for convolution operation in CNNs. If the number of partial products summation is large enough, throughput of the proposed MAC on a systolic array will be the clock frequency divided by 1 and 4 for 16-bit and 32 bit mode computation.

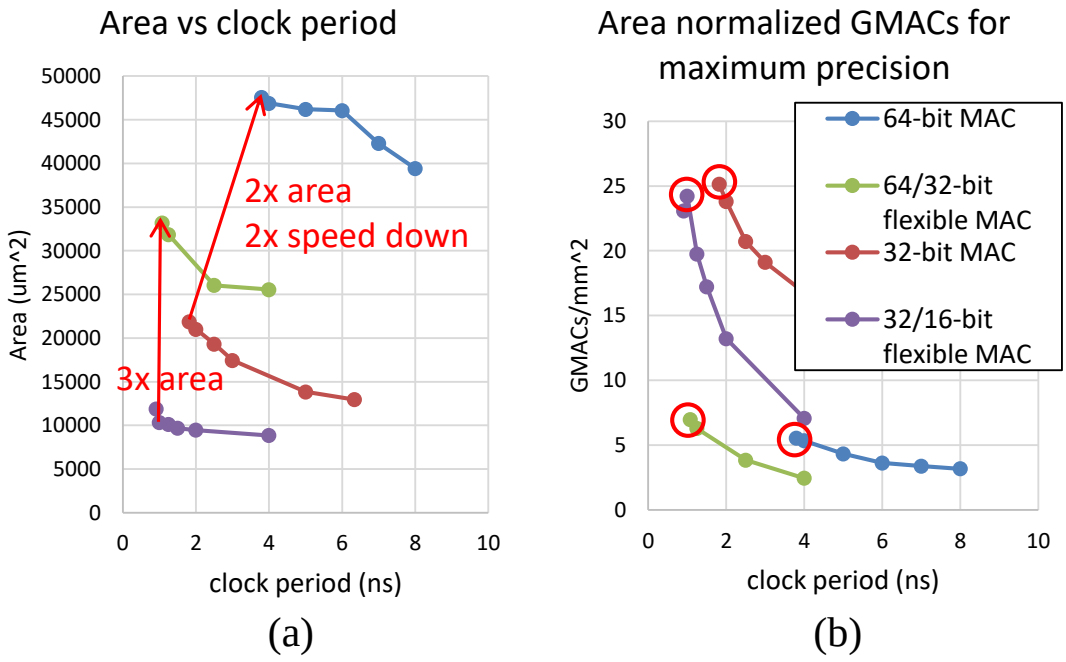


Figure 3.7: (a) Area vs. clock period (b) Area normalized GMACs for maximum precision mode.

3.4.3 Scalability for the proposed MAC

The proposed MAC is designed and implemented with Synopsys 28nm PDK (supply voltage of 1.05V). Design space exploration has been performed for 64-bit, 32-bit traditional MAC, 64/32-bit and 32/16-bit flexible MAC. Synopsys DesignWare library is used since it provides area-delay optimized results for traditional MAC unit for fair comparison. Being too flexible like 32/8-bit or 64/16-bit flexible MAC shows poor results than 32/16-bit or 64/32-bit flexible MAC respectively in terms of area normalized throughput for maximum precision mode, thus, the results are not reported in this study. Area normalized figure of merits are valid since the MAC is used in a systolic array where large number of MACs can be placed closely.

Figure 3.7 shows the implementation results after the place and route. The optimal design point is chosen based on the results of area normalized Giga MAC operations per

Table 3.1: 28nm implementation results

	64-bit	32-bit	16-bit	64/32-bit MAC		32/16-bit MAC	
precision (bit)	64	32	16	64	32	32	16
clock period (ns)	3.8	1.82	1.5	1.08		1.0	
area (um^2)	48K	22K	6K	33K		10K	
GMACS/mm^2	5.5	25.1	111	7.0	**27.9**	24.2	**96.8**
Systolic array ($1mm^2$) num_row	4	8	16	4		10	
Systolic array ($1mm^2$) num_col	5	6	10	8		10	

Second (GMACS/mm^2). Design points which correspond with the red circles in Figure 3.7(b) are chosen for further comparison.

Table 3.1 summarizes the implementation results. 16-bit conventional MAC is also implemented as a reference. As shown in this table, area of 64/32-bit flexible MAC is increased by 3.3x compared to 32/16-bit flexible MAC while operating frequency remains almost the same due to simple re-timing in configuration MAC unit. This results in 3.4x degradation in area normalized Giga MAC operations per Second (GMACS).

For traditional MAC, area and the clock period of 64-bit MAC are increased by 2.2x and 2.1x compared to 32-bit MAC respectively resulting 4.5x degradation in area normalized throughput. As a result, 64/32-bit flexible MAC shows better area normalized throughput (7.0 GMACS/mm^2) than traditional 64-bit MAC (5.5 GMACS/mm^2) for 64-bit mode. 32-bit MAC and 32/16-bit flexible MAC shows almost the same results for 32-bit mode. If the proposed flexible MAC is used for low precision mode with DPS, performance will increase since the proposed MAC will run at a lower latency. Also the flexible MAC has similar performance with traditional MAC for maximum precision mode.

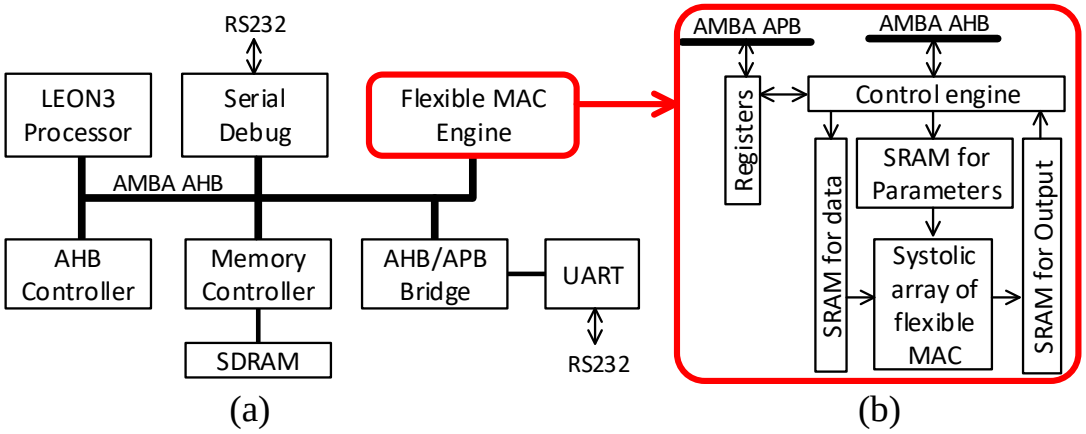


Figure 3.8: (a) Overall system architecture and (b) flexible MAC engine.

3.5 Performance Evaluation

In this section, the performance of the coupled algorithm and hardware approach is presented.

3.5.1 Systme Architecture

Figure 3.8(a) shows an example overall system architecture and Figure 3.8(b) shows the proposed hardware IP. Training runs with software on the processor and the hardware IP offloads the convolution tasks. While training, the software configures the proposed hardware by setting register values and the control engine in IP orchestrates data movement. Systolic array architecture is adopted since it can utilize the flexible MAC in an area efficient way. General matrix-matrix operation (GEMM) is performed as shown in Figure 3.9. The number of cycles/GEMM can be obtained by:

$$\frac{\#cycles}{GEMM} = n \times (num\_row + num\_col + kernelsize - 2) + added\_latency \quad (3.1)$$

,where n is the number of cycles required for all partial products generation in configurable MAC unit, num_row and num_col are dimension of systolic array, $kernelsize$

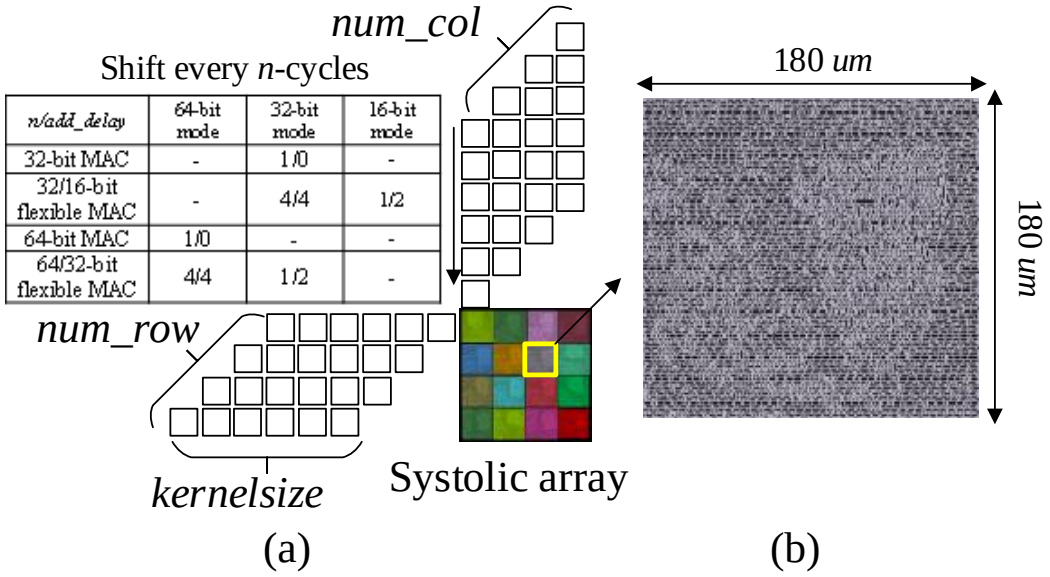


Figure 3.9: (a) Systolic array with MAC (b) 28nm layout of 64/32-bit flexible MAC.

represents the kernel size and $added\_latency$ is the added latency for the flexible MAC cases. And the number of GEMM operation is obtained by:

$$\#GEMM = ceil(\frac{\#batch}{num\_row}) \times ceil(\frac{\#kernel}{num\_col}) \times out\_size \tag{3.2}$$

, where $\#batch$ is the batch size, $\#kernel$ is the number of kernels in a convolution layer, out_size is widthheight of the output layer. The utilization of systolic array per single GEMM operation can be obtained by:

$$Utilization/GEMM = n \times kernelsize \div (\frac{\#cycles}{GEMM}) \tag{3.3}$$

To evaluate the proposed design for deep neural network training, 1 mm^2 die area is assumed for the systolic array. The dimension and the clock period for each systolic array is shown in the Table 3.1. 32-bit MAC for Lenet and 64-bit MAC for the modified Alexnet are used as baseline cases. 16-bit and 32-bit MAC for Lenet and the modified

Table 3.2: GMACS for convolutional layers in Lenet (# batch: 64)

# kernels	kernel size	32-bit baseline	16-bit w/o DPS (32-bit)	32/16-bit MAC w/ DPS (16-bit)
20	25	14.8	12.5 (0.8x)	50.7 (**3.4x)**
50	500	23.8	21.9 (0.9x)	87.9 (**3.7x)**

Table 3.3: GMACS for convolutional layers in modified Alexnet (# batch: 64)

# kernels	kernel size	64-bit baseline	32-bit w/o DPS (64-bit)	64/32-bit MAC w/ DPS (32-bit)
96	363	4.9	7.1 (1.4x)	28.6 (**5.8x)**
256	1200	5.1	7.3 (1.4x)	29.3 (**5.7x)**
384	2304	5.2	7.3 (1.4x)	29.4 (**5.6x)**
384	1728	5.2	7.3 (1.4x)	29.4 (**5.6x)**
256	1728	5.1	7.3 (1.4x)	29.4 (**5.7x)**

Alexnet are included respectively just for comparison purpose assuming traditional MAC is tailored for good-enough precision found from the DPS algorithm. This is an ideal case since tailored conventional MAC cannot offer flexible precision mode. It assumes that the optimum precision is already known as a priori. For the proposed flexible MAC, 32/16-bit MAC and 64/32-bit MAC for Lenet and modified Alexnet are used respectively.

3.5.2 Evaluation

Table 3.2 and 3.3 show the performance summary. First, throughput increases as kernel size increases since it can maximize the utilization of a systolic array. This is due to the fact that # cycles per GEMM becomes dominated not by size of the systolic array but by kernel size as kernel size becomes larger. Figure 3.10 shows this trend. Also the utilization of systolic array for high precision MAC is larger than that for low precision MAC given kernel size.

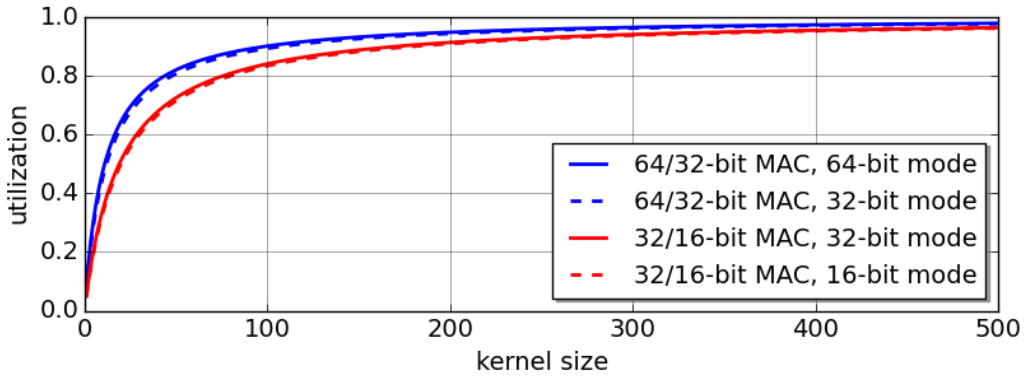


Figure 3.10: Systolic array utilization

The number ($num\_row \times num_col$) of high precision MACs in given systolic array area is smaller than that of low precision MACs. Therefore, the utilization of a systolic array with high precision MAC becomes higher than that with low precision MAC.

Second, combined use of DPS and flexible MAC for the modified Alexnet achieves better performance than that for Lenet. This is because the flexible MAC is more scalable as seen in the previous section. It is desirable considering the fact that acceleration of training is actually needed for larger and deeper neural network where training time is considerable.

Third, the proposed 64/32-bit flexible MAC with DPS algorithm for the modified Alexnet can speed up training up to 5.8x compared to baseline 64-bit fixed point MAC. This is promising result since the most of the training time is spent on convolutional layer [72]. Also, overhead of the proposed algorithm is not huge since it uses already-calculated train loss and overflow detection which is built-in function in the hardware. One-time check of overflow registers is sufficient after forward/backward passes per iteration.

Figure 3.11 shows the relative energy per task normalized with high precision MAC energy (32-bit and 64-bit MAC for Lenet and Alexnet respectively). When the proposed MAC and DPS are used at the same time, energy per task is reduced by 45% and 69% for Lenet and modified Alexnet respectively compared to baseline high precision fixed point MAC. When the flexible MAC is used for high precision mode only, energy consumption

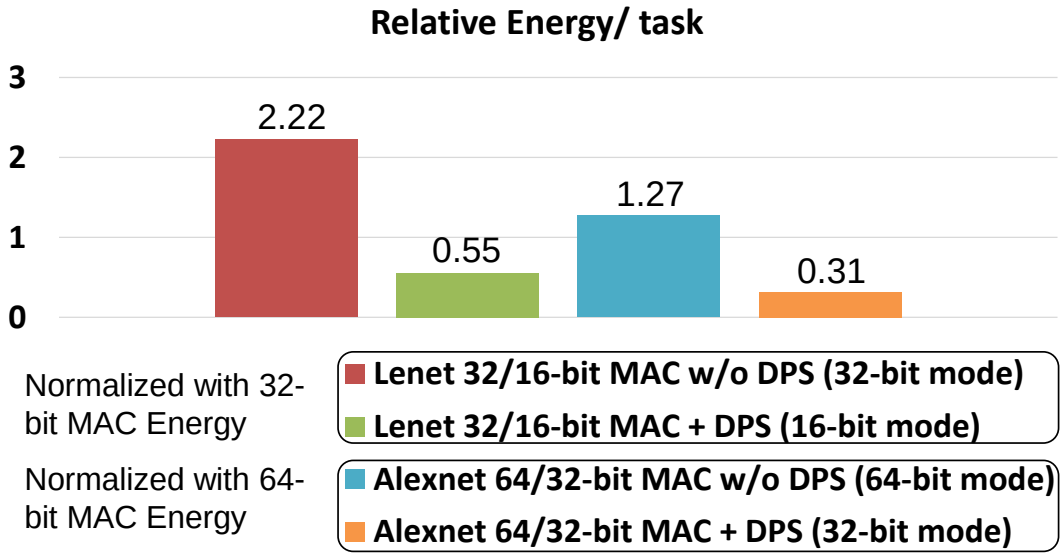


Figure 3.11: Relative energy/task

of the flexible MAC is higher than that of high precision fixed point MAC since the MAC operates at a higher clock frequency. Note that 'Lenet DPS 16-bit MAC' and 'Alexnet 32-bit MAC' cases show ideal results which assume tailored MAC for the final good enough precision from DPS algorithm. Even though low precision computation of the proposed flexible MAC consumes more energy than low precision tailored MAC, the benefit of using precision flexible MAC is huge since it enables actual speeding up of training.

3.6 Summary

This section proposed a coupled algorithm-hardware co-design approach to accelerate training of deep neural network. An efficient DPS algorithm is proposed to find good enough precision while maintaining accuracy for convolutional neural network training. A flexible MAC unit is presented for efficient hardware realization of the proposed algorithm that provides increased throughput at lower precision. The proposed coupled approach significantly reduces training time by up to 5.7x while consuming 31% of MAC energy compared to conventional fixed-point approach on the modified Alexnet for image style recognition

on the Flickr style images. As fast training of bigger and deeper network is becoming increasingly important for deep learning, the proposed approach can provide an efficient platform for hardware accelerated training.

CHAPTER 4

LOW PRECISION TRAINING OF RECURRENT NEURAL NETWORKS

4.1 Introduction

Recurrent neural networks (RNNs) have shown great potentials for various machine learning tasks such as machine translation [10, 11, 12], speech recognition [13, 14], and even games [15, 16], with the ability of learning long term dependencies. Long short term memory (LSTM) [18] and gated recurrent unit (GRU) [19] have enabled such achievements.

Meanwhile, there is a growing interest towards deploying neural networks onto mobile edge devices for on-chip training and inference with the recent advances in internet-of-things [73]. However, huge computational demand for their training makes it hard to deploy neural networks on resource constrained edge devices. Moreover, RNNs are known to require more computation to train than feed-forward networks due to the vanishing and exploding gradient problems [20]. Therefore, the most critical goal of designing an on-chip system for RNN training is to reduce the energy and time required for training.

To address this challenge, various hardware approaches have been proposed for efficient computation of RNN models. For instance, Long et al. [21] have introduced a programmable RNN using Resistive Random Access Memory (ReRAM). Although it achieves training speed-up with less energy, the device variation of ReRAM significantly degrades the system performance. Ott et al. [22] have shown that limited numerical precision can be applied in RNNs to reduce computation by quantizing weights and eliminating multiplications in the forward pass. However, this work has applied low precision only in the forward pass, requiring floating point precision for updating the parameters during backward propagation.

This chapter presents limited precision training of RNNs for both forward pass and

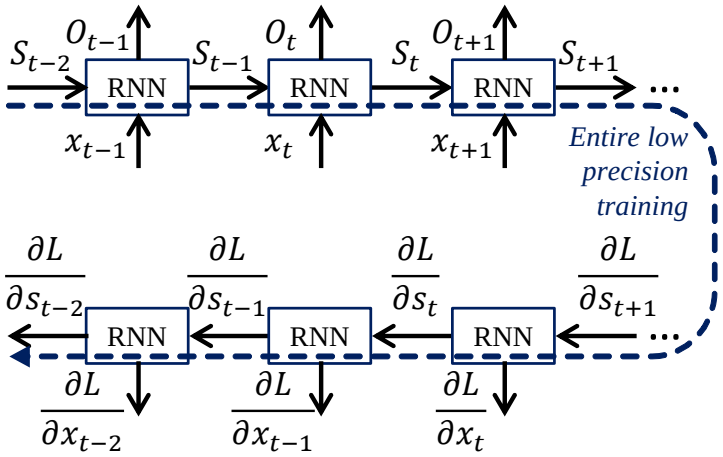


Figure 4.1: Limited numerical precision training of RNN for forward and backward pass.

backward pass (Figure 4.1. GRU is used as a target RNN and use the dynamic fixed point format [69] as a candidate numeric format. GRU is used with batch normalization [43] on input sequences for human activity recognition with the KTH dataset [74]. The simulation results show that batch normalization improves speed of training with low precision as well as floating point precision.

Interesting observations found in this chapter include:

1. Batch normalization known to be effective in CNNs is essential even for RNNs training.
2. 64-bit fixed point is not sufficient for training RNNs.
3. The overflow rates, which determine decimal points for dynamic fixed points, should be carefully controlled to enable successful low precision training.
4. Stochastic rounding achieves superior results than other options.
5. Gradient accumulation for the weights if more important than any other path,
6. Piecewise linear activation together with stochastic rounding works pretty well.

Low precision multiplier and accumulator (MAC) with linear-feedback shift register (LFSR) is implemented with 28nm Synopsys PDK for energy and performance analysis. Implementation results show that low precision hardware is 4.7x faster, and energy per task is up to 4.55x lower than that of floating point hardware.

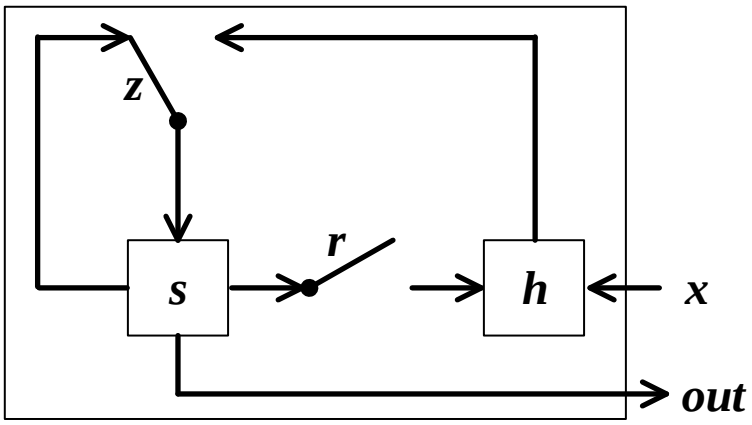


Figure 4.2: GRU gating [19]

4.2 Gated Recurrent Unit Background

4.2.1 Gated Recurrent Unit

GRU has one hidden state (s) and two gates, a reset gate (r), and an update gate (z) as shown in Figure 4.2. The current state(s_t) is combined with the previous state and the candidate state (h). The update gate determines portion of the previous state (s_{t-1}) and the current candidate state for updating the state. The candidate state is a combination of the input (x) and the previous state where the reset gate determines how to transfer the previous state values for the candidate. The output (out) is a function of the current state.

All the gates and the candidate state are calculated based on the previous state, and the current input and learned parameters are used to calculate the gates and candidate. Associated equations are given by:

$$z = \sigma(x_t \cdot U^z + s_{t-1} \cdot W^z) \tag{4.1}$$

$$r = \sigma(x_t \cdot U^r + s_{t-1} \cdot W^r) \tag{4.2}$$

$$h = tanh(x_t \cdot U^h + (s_{t-1} \circ r) \cdot W^h) \tag{4.3}$$

$$s_t = (1 - z) \circ h + z \circ s_{t-1} \tag{4.4}$$

$$\hat{y}_t = softmax(s_t \cdot V) \tag{4.5}$$

where x_t is the input vector at time step t, U^z, U^r, and U^h are the input to hidden weight matrices, W^z, W^r, and W^h are the hidden to hidden weight matrices, V is the hidden to output weight matrix, s_{t-1} is the state at $t-1$, and $\hat{y}_t$ is the estimated output vector at time t. Here, bias terms for z, r, h and $\hat{y}_t$ are omitted for brevity. σ is a sigmoid function and $\circ$ is element-wise matrix multiplication.

For batch training, x_t is an n_batch-by-n_in matrix, where n_batch is the mini batch size and n_in is the size of a single input at time t. U^z, U^r, and U^h are n_in-by-n_hidden matrices, where n_hidden is the hidden state size. W^z, W^r, and W^h are n_hidden-by-n_hidden matrices, and z, r, h, and st are n_batch-by-n_hidden matrices. V is an n_hidden-by-n_out matrix, where n_out is the size of the output, and $\hat{y}_t$ is an n_batch-by-n_out matrix. Categorical cross entropy is used for loss function which is given by:

$$L(y_t, \hat{y}_t) = -\frac{1}{n\_out} \sum_{i=1}^{n\_out} y_t^i \log(\hat{y}_t^i) \tag{4.6}$$

where y_t^i is the ground truth at time t.

4.2.2 Backpropagation Through Time

When calculating gradients with respect to weight matrices, unrolled network through time is used. Figure 4.3 shows all basic equations of BPTT for GRU. Suppose, t = 1, ..., T, where T is the last time step for a given input. The gradient w.r.t. the last state ($\partial L / \partial s_T$)

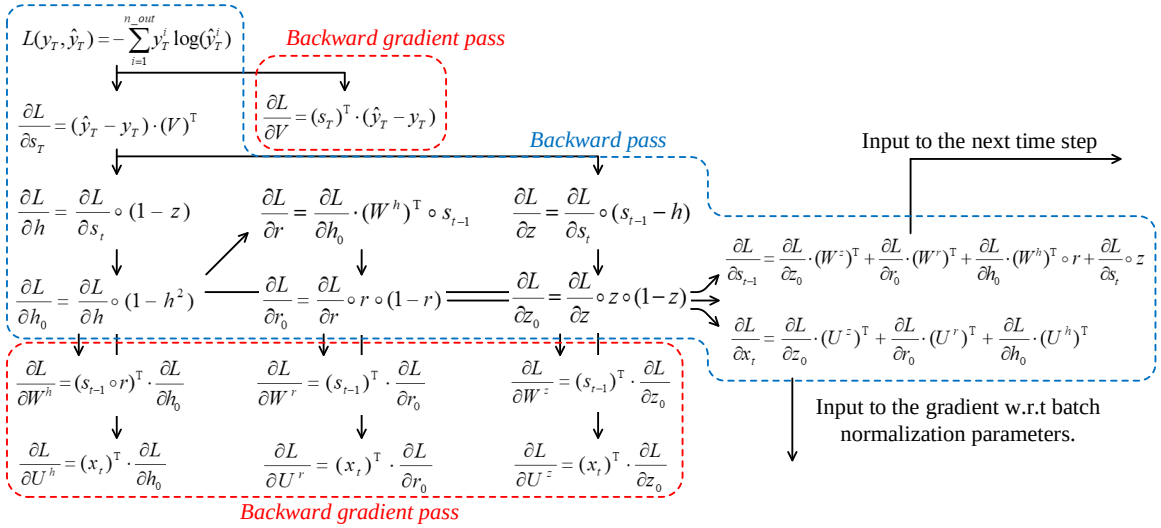


Figure 4.3: BPTT for GRU at single time step t. Intermediate terms (r, z and h) for each time step are stored during forward pass and used during backward pass for BPTT. Gradient accumulation is not shown for brevity.

is first calculated, and then fed into the first backward propagation through time. Series of the gradient terms can be calculated by the chain rule as shown in Figure 4.3.

Once $\partial L/\partial s_T$ calculation is done, $\partial L/\partial z$ and $\partial L/\partial h$ can be calculated. $\partial L/\partial r$ cannot be directly calculated since it is not the direct function of s_t. z_0, r_0 and h_0 are defined to be the terms before the activation of z, r, and h, respectively.

$$z_0 = x_t \cdot U^z + s_{t-1} \cdot W^z \tag{4.7}$$

$$r_0 = x_t \cdot U^r + s_{t-1} \cdot W^r \tag{4.8}$$

$$h_0 = x_t \cdot U^h + (s_{t-1} \circ r) \cdot W^h \tag{4.9}$$

Sigmoid and tanh functions are differentiable, thus, $\partial L/\partial z_0$ and $\partial L/\partial h_0$ can be eas-

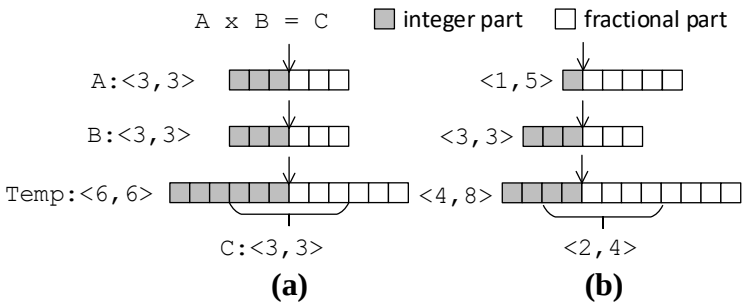


Figure 4.4: Multiplication examples for (a) conventional fixed point and (b) dynamic fixed point.

ily calculated. Once $\partial L/\partial h_0$ is obtained, $\partial L/\partial r$ and $\partial L/\partial r_0$ can be obtained. Finally $\partial L/\partial U^z$, $\partial L/\partial U^r$, $\partial L/\partial U^h$, $\partial L/\partial W^z$, $\partial L/\partial W^r$, $\partial L/\partial W^h$, $\partial L/\partial s_{t-1}$, and $\partial L/\partial x$ can be calculated. $\partial L/\partial s_{t-1}$ is fed into the next backward time step, and $\partial L/\partial x$ is used to calculate the gradients w.r.t. the batch normalization parameters.

Each term can be calculated with the General matrix-matrix operation (GEMM) or element-wise multiplication. Multiplier and accumulator (MAC) is a general choice for the GEMM operation, and it gives higher precision for intermediate products. Thus, applying quantization right before the assignment for each term is reasonable for low precision training in RNNs.

$\partial L/\partial U^z$, $\partial L/\partial U^r$, $\partial L/\partial U^h$, $\partial L/\partial W^z$, $\partial L/\partial W^r$, $\partial L/\partial W^h$, and $\partial L/\partial V$ calculation (enclosed with red dotted line in Figure 4.3) are referred to as "backward weight pass" and the other pass (enclosed with blue dotted line in Figure 4.3) as "backward hidden pass". These notations will be used in later section to divide regions for stochastic rounding.

4.3 Low Precision Training Framework

4.3.1 Quantization

Dynamic fixed point [69] is utilized as a target numeric format. As shown in Figure 4.4, dynamic fixed point allows several decimal points instead of a single global one. Compared

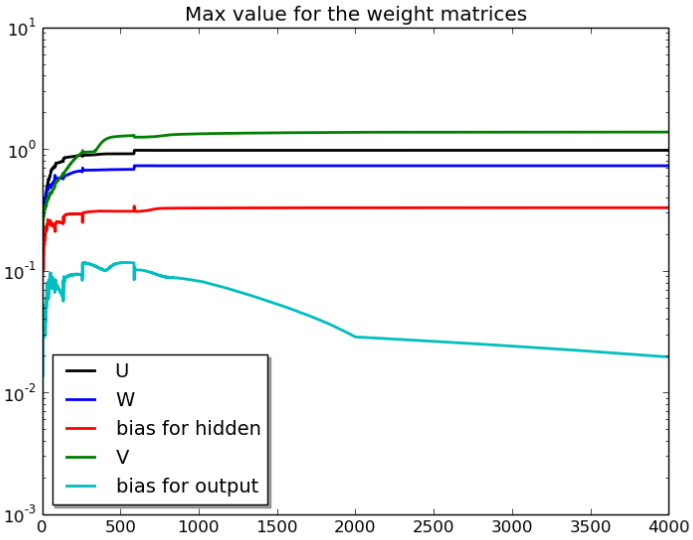


Figure 4.5: Maximum values for different weight matrices during training.

to feedforward neural networks where the most commonly used nonlinearity activation function is a rectified linear unit, in RNNs, the state values are bounded by 1 and the gate values are between 0 and 1 for forward pass since the activation functions (σ and $tanh$) limit the values. However, weight matrices and the intermediate terms in the backward pass in RNNs can have large variances. Figure 4.5 shows maximum values for the weight matrices for GRU with human activity data set. The statistics are different for the weight matrices, and gradient terms are not bounded during BPTT. Thus, applying the dynamic fixed-point format seems to be a natural choice for fixed point arithmetic.

Quantization is performed for every intermediate term for low precision training emulation. For the forward pass, weight matrices $U^z, U^r, U^h, W^z, W^r, W^h$, and V are first quantized as shown in Figure 4.6. $Q_{U^z}, Q_{W^z}, Q_{U^r}, Q_{W^r}, Q_{U^h}, Q_{W^h}$, and Q_V are the quantization functions for $U^z, U^r, U^h, W^z, W^r, W^h$, and V respectively. Those are distinct quantization functions making dynamic fixed point format with different decimal point. Quantization functions for the weight matrices is referred to as "forward weight pass" quantization.

Those weights can be quantized either right before the forward pass or during update.

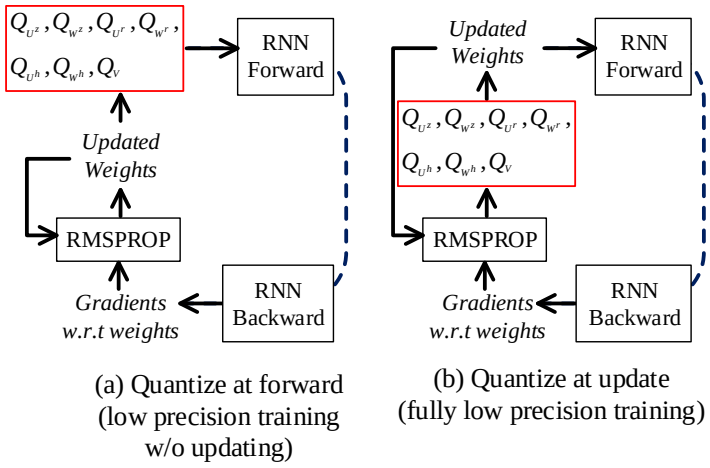


Figure 4.6: Two different emulations for weight matrices update.

First option is to quantize the weights at the update phase. In this case, newly computed values are quantized first, and then the weights are replaced with the new quantized values. This emulates fully low precision training even for the update process. Second option is to update weights on the full precision weights and quantize weights only at the forward pass. This emulates low precision training only for the forward/backward passes, not for the update process. Since the update process takes very little time compared to the entire training process, one might use full precision for the update step. These two options will be studied in later section.

Once the weight matrices are quantized, x_t, z, r, h, s_t, and $\hat{y}_t$ are quantized. x_t is the quantized output from batch normalization as follows:

$$x_t = Q_x(\gamma(a_t - mean(a_t))/\sqrt{var(a_t)} + \beta) \tag{4.10}$$

where at is the original input at time t, and γ and β are the parameters for the batch normalization. Q_x is a quantization function for x_t. Likewise, z, r, h, s_t, and $\hat{y}_t$ are also quantized right before the assignment as follows:

$$z = Q_s(z_0) \tag{4.11}$$

$$r = Q_s(r_0) \tag{4.12}$$

$$h = Q_s(h_0) \tag{4.13}$$

$$s_t = Q_s((1 - z) \circ h + z \circ s_{t-1}) \tag{4.14}$$

$$\hat{y}_t = Q_s(softmax(s_t \cdot V)) \tag{4.15}$$

where Q_s is a quantization function for $\{z, r, h, s_t, \hat{y}_t\}$. z, r, h, s_t, and $\hat{y}_t$ use the same quantization function since those values are bounded either by [-1, 1] or [0, 1]. Q_x and Q_s are also distinct dynamic fixed point functions where each function has different decimal point. Q_x and Q_s are referred to as "forward hidden pass" quantization.

Similarly, for the backward pass, each intermediate term and final gradient term in Figure 4.3 has its own quantization function before assignment.

4.3.2 Rounding Operations

Three different rounding options are considered. The first option is bit truncation. Bit truncation for signed two's complement fixed point format is essentially a round down operation as follows:

$$Q(x) = \lfloor x \cdot 2^{\mathrm{FL}} \rfloor / 2^{\mathrm{FL}}$$

where FL is the fractional part bit-width. The second option is a round to nearest operation as follows:

$$Q(x) = \lfloor (x + \epsilon/2) \cdot 2^{\mathtt{FL}} \rfloor / 2^{\mathtt{FL}}$$

where the smallest positive number ϵ is $2^{-\mathtt{FL}}$. The third option is stochastic rounding, which is computed by:

$$Q(x) = \begin{cases} \lfloor x \cdot 2^{\mathtt{FL}} \rfloor / 2^{\mathtt{FL}}, w.p = \lceil x \cdot 2^{\mathtt{FL}} \rceil - x \cdot 2^{\mathtt{FL}} \\ \lfloor (x + \epsilon) \cdot 2^{\mathtt{FL}} \rfloor / 2^{\mathtt{FL}}, w.p = x \cdot 2^{\mathtt{FL}} - \lfloor x \cdot 2^{\mathtt{FL}} \rfloor \end{cases}$$

In this mode, a rounding operation is performed stochastically based on how close x is to the quantized values between $\lfloor x \cdot 2^{\mathtt{FL}} \rfloor / 2^{\mathtt{FL}}$ and $\lfloor (x + \epsilon) \cdot 2^{\mathtt{FL}} \rfloor / 2^{\mathtt{FL}}$.

After quantization, the values are clipped to $[-2^{\mathtt{IL}-1}, 2^{\mathtt{IL}-1-\epsilon}]$, where IL is the integer part bit-width. The overflow rate is determined by counting occurrence of clipping over the total number of quantization.

4.3.3 Decimal Point Search for Dynamic Fixed Point

A dynamic fixed point format is used as a target numerical format. Each quantization function object has its own $<$ IL, FL$>$ parameter, where IL is integer part bit-width and FL is fractional part bit-width. The problem is to find the optimal decimal point for each quantization object during training. A simple algorithm is proposed to determine the optimal decimal point for each quantization object. Essentially, the algorithm runs on top of the training process. Once a single forward/backward/update step is done, the decimal point search function is called.

Details of this heuristic are shown in Algorithm 2. The first step is to initialize decimal point for every quantization object ($Q.\mathtt{initialize}(tl, fl)$) in $Q|_l ist$ based on the initial total bit-width (tl) and fractional part bit-width (fl). The internal $mode$ variables in quantization objects are initialized to zero ($Q.\mathtt{setMode}(0)$). Also, global $search\_mode$

Algorithm 2 Decimal Point Search Algorithm

Require: $threshold$: overflow threshold, Q_list: Quantization objects list, tl: Initial total bit width, fl: Initial fractional part bit width.

for Q **in** Q_list
    Q.initialize(tl, fl)
    Q.setMode(0)
$search\_mode \leftarrow$ IL_DEC
repeat
    trainOnestep()
    switch $search\_mode$ **do**
        case IL_DEC
            $sum\_mode \leftarrow 0$
            for Q **in** Q_list
                if Q.getOverflow() $> threshold$ **then**
                    Q.setIL(+1)
                    Q.setMode(1)
                else if Q.getMode() == 0 **then**
                    Q.setIL(-1)
                $sum\_mode+ = Q$.getMode()
            if $sum\_mode == \texttt{size}(Q\_list)$ **then**
                $mode \leftarrow$ IL_INC
        case (IL_INC)
            for Q **in** Q_list
                if Q.getOverflow() $> threshold$ **then**
                    Q.setIL(+1)
until training converged

is initialized to IL_DEC. Once a single training step (`trainOnestep`()) is done, internal overflow rate for every quantization object is set as a result of low precision training emulation. If $search\_mode$ is IL_DEC, integer part bit-width is decreased by 1 (Q.`setIL`(-1)). This is done if the overflow rate has not exceeded the given $threshold$ since the training is initiated. As the integer part bit-width continues to decrease, the overflow rate will eventually exceed the $threshold$. If the overflow rate is greater than $threshold$, the integer part bit-width is increased by one (Q.`setIL`(+1)) and set mode to one (Q.`setMode`(1)). If every quantization object has ever experienced exceeding of $threshold$, $search\_mode$ is changed to IL_INC. In the IN_INC mode, only increasing the integer part bit-width is allowed when the overflow rate is greater than $threshold$.

4.4 Experimental Results

4.4.1 Simulation Setup

GRU is used for human activity recognition with KTH dataset. The dataset contains 25 persons, 4 scenes and 6 activities (boxing, hand clapping, hand waving, jogging, running, and walking). The total time step T is 200. Space time interest points (STIP) [17] is utilized as a local feature extractor, which produces input size (n_in) of 168. During training, an n_batch-by-n_in matrix (x_t) is fed into GRU at time t, and an n_batch-by-n_out matrix $\hat{y}\_t$ is generated. Even though $\hat{y}_t$ is generated at every time t, the last output ($\hat{y}_T$) is used to calculate the loss.

In the simulation, n_hidden is 40 and n_batch is 42. As the total training size is 168 (dataset with 7 persons), one epoch training is equivalent to four iterations.1,000 epochs are used for training. RMSProp optimizer is used for the update algorithm, and clip the gradients with 1. BPTT truncation is also used with step 20. The program is implemented with Theano library in python [75]. 24 bit total bit-width, 0.005 for overflow rate threshold, round to nearest, and fully low precision training is used by default for low precision training emulation unless otherwise noted.

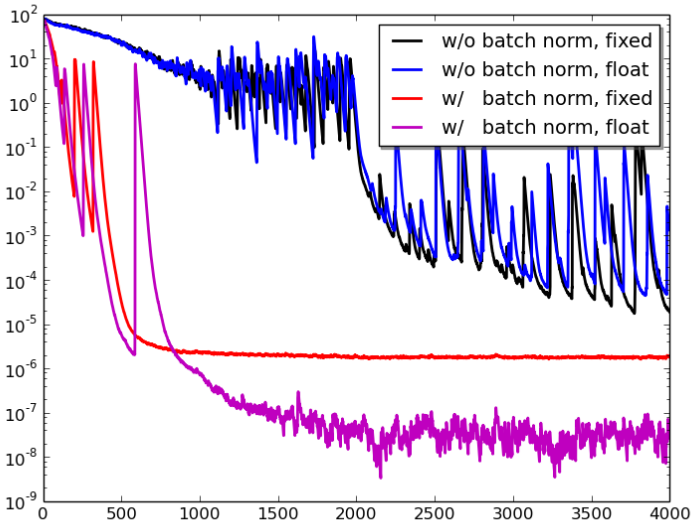


Figure 4.7: Loss with and without batch normalization. < bit-width: 24, round to nearest, overflow rate: 0.005, and fully low precision training>

4.4.2 Batch Normalization

Batch normalization has been introduced first in convolutional neural networks (CNNs). It allows reducing internal covariance shift during training, thus, makes training faster. Recent studies showed that batch normalization is effective even in RNNs [76, 77]. Batch normalization is used for the input sequences in low precision training and compared with the baseline setting (without batch normalization).

Figure 4.7 shows training exponential moving average loss vs iteration with and without batch normalization on the input sequences for 24 bit dynamic fixed points and floating points. Exponential moving average loss is used to show the results since the raw loss data is somewhat noisy due to irregular loss surface. Note that the batch normalization is beneficial for training with low precision as well as floating point precision. Thus, batch normalization is always applied for low precision training in the following sections. Note floating point precision is used for batch normalization parameters ($\gamma and \beta$) since batch normalization is essential to enable low precision training. Parameters needed for batch normalization are just two (γ and β) and the batch normalization process takes small por-

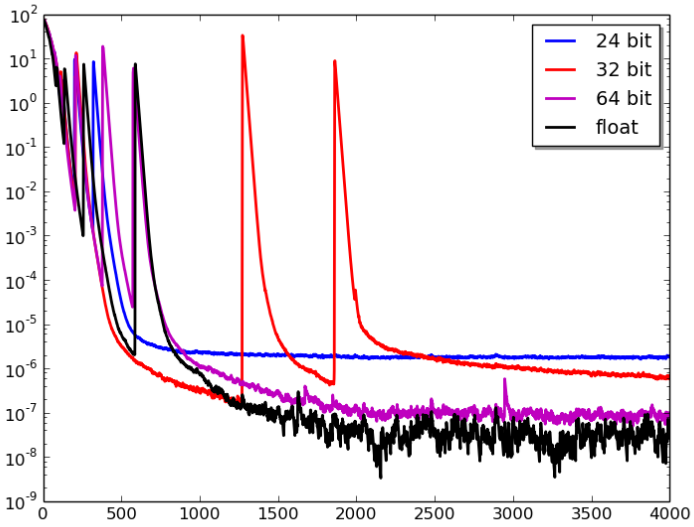


Figure 4.8: Loss with 24-bit, 32-bit and 64-bit dynamic fixed points and floating points. < round to nearest, overflow rate: 0.005, and fully low precision training>

tion among the entire training process.

4.4.3 Precision

Study of impact of various numerical precisions on low precision training is presented. Figure 4.8 shows training results for 24-bit, 32-bit, and 64-bit dynamic fixed point, and floating point as a reference. As shown in this figure, as precision becomes higher, resulting loss becomes lower. It is interesting to note that for 32-bit dynamic fixed point, sudden peaks are observed during training even at higher iterations making harder to converge. Also note that even 64-bit fixed point does not seem to be sufficient for training RNNs, while 64-bit fixed points is known to be sufficient for training CNNs [78].

4.4.4 Overflow Rate

As discussed in the previous section, the overflow rate threshold is used to determine decimal points for all quantization objects. Figure 4.9 shows several experiments for various overflow rate thresholds for 24-bit dynamic fixed point. 24-bit precision is used since it is

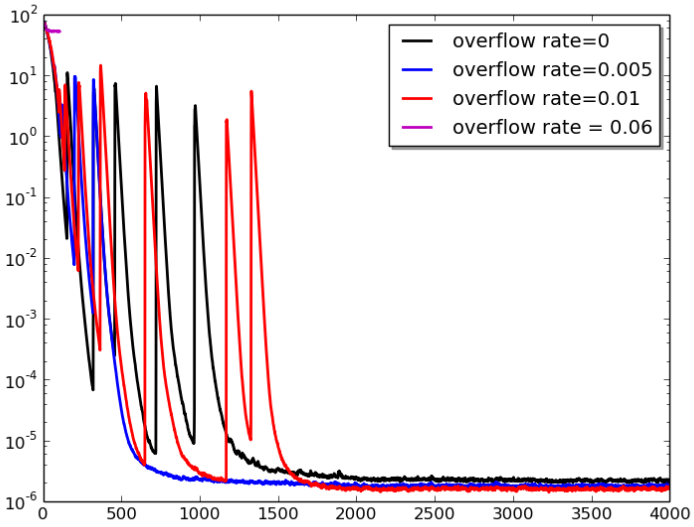


Figure 4.9: Loss with overflow rate threshold = 0, 0.05, 0.01 and 0.06. < bit-width: 24, round to nearest, fully low precision training>

easy to show and differentiate effects coming from various hyper parameters even though it does not produce floating point-like training.

As shown in Figure 4.9, a high threshold greater than 0.01 leads training unstable. Also, a threshold value of 0 does not give the optimal point. A threshold around 0.005 gives stable results and use 0.005 as default threshold for the overflow rate.

4.4.5 Rounding

In this section, various rounding options are studied. Figure 4.10 shows simulation results for various rounding options including bit truncation, round to nearest, and stochastic rounding for 24 bit dynamic fixed point format. Bit truncation shows poor results among the three possible rounding options. Round to nearest results are in the middle between bit truncation and stochastic rounding. Stochastic rounding shows comparable results with floating point precision even with 24-bit fixed point precision. It is noteworthy since 64-bit round to nearest does not produce comparable result with floating point precision.

Regions are divided and stochastic rounding is applied separately. As discussed in

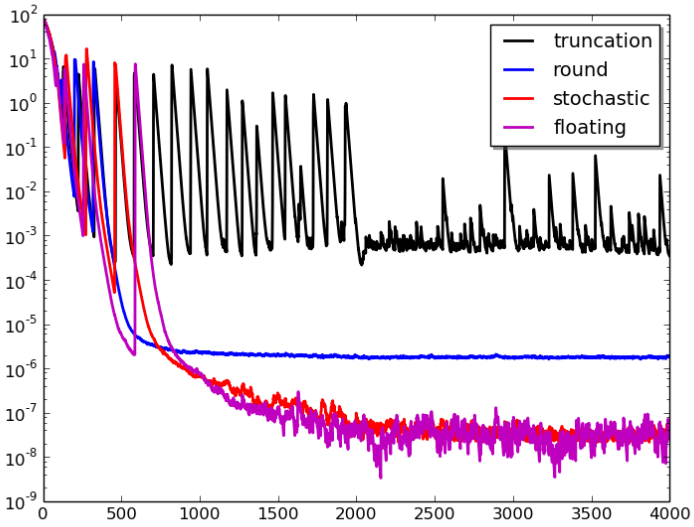


Figure 4.10: Loss with bit-truncation, round to nearest, stochastic rounding and floating point as a reference. < bit-width: 24, overflow rate: 0.005, and fully low precision training>

earlier section, "forward hidden", "forward weight", "backward hidden", and "backward weight" passes are used to divide the region. Figure 4.11 and 4.12 show simulation results with stochastic rounding in various regions.

In Figure 4.11, stochastic rounding is applied in a single region to understand what is the most influential pass for stochastic rounding. Applying stochastic rounding on "hidden passes" ("forward hidden" and "backward hidden") does not impact much on the training. Applying stochastic rounding on "weight passes" can have more impact on training than applying on "hidden passes". Stochastic rounding can be most effective when applied on accumulating the gradients ("backward weight").

In Figure 4.12, stochastic rounding is applied in a group where each group contains two passes. Stochastic rounding in "forward" or "hidden" does not give any benefit of using stochastic rounding. Stochastic rounding in "backward" or "weight" gives some benefit of using stochastic rounding though it does not reach at the level of full stochastic rounding. Stochastic rounding needs additional resources when implemented in actual hardware, thus, one might use stochastic rounding only for accumulating gradients in resource limited

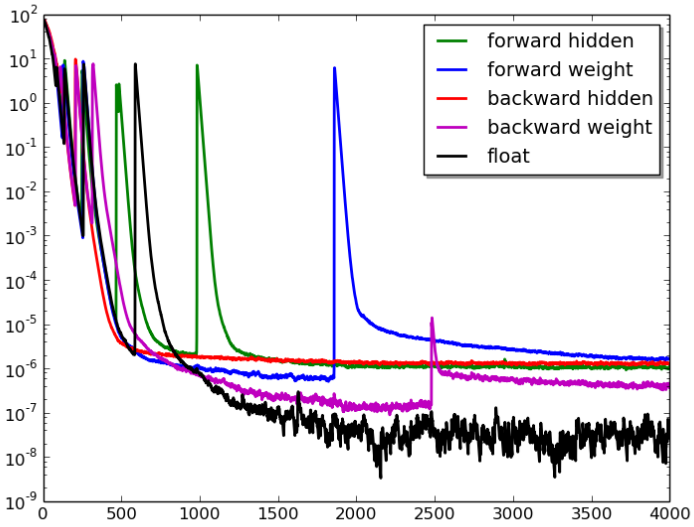


Figure 4.11: Loss with stochastic rounding in various regions. < bit-width: 24, round to nearest, overflow rate: 0.005, and fully low precision training >

environments.

4.4.6 Fully Low Precision vs Partial Low Precision Training

In this section, the effect of partial low precision training is studied compared to fully low precision. Figure 4.13 shows simulation results with fully low precision, forward/backward low precision with high precision weight update, and forward only low precision. As shown in this figure, maintaining high precision for updating weight matrices does not help much for training. It is almost the same with fully low precision training. When applying low precision only for the forward pass, high precision for backward and update does not improve training.

4.4.7 Piecewise Linear Activation

Since sigmoid and tanh functions have to compute exponential function, one might consider applying a piecewise linear activation function to reduce computation. The hard_sigmoid and ultra_fast_sigmoid functions are used in Theano library. Essentially hard_sigmoid is

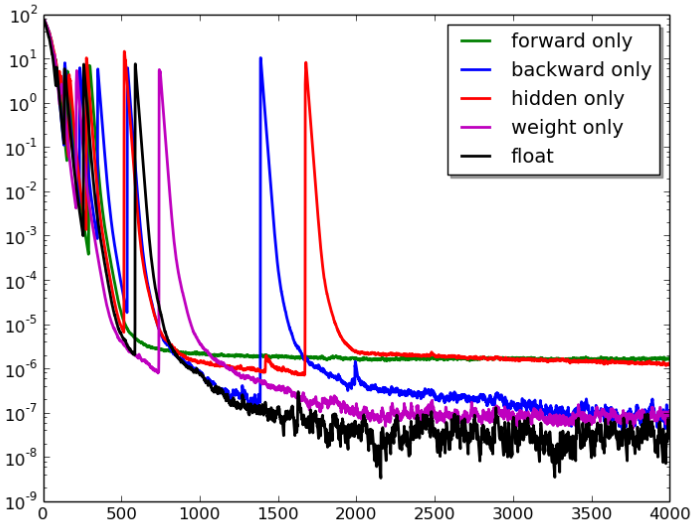


Figure 4.12: Loss with stochastic rounding in various regions. < bit-width: 24, overflow rate: 0.005, and fully low precision training>

three-part linear approximation and ultra_fast_sigmoid is five-part linear approximation. Piecewise linear function for tanh is defined as:

$$piecewise\_tanh(x) = 2 \cdot piecewise\_\sigma(2x) - 1$$

where $piecewise\_\sigma$ can be either the hard_sigmoid or ultra_fast_sigmoid function. Piecewise linear activations are only used for forward pass. During the backward pass, the gradients obtained from non-piecewise linear function are used instead of using the actual gradient since the gradient for hard_sigmoid is 0.2 between -2.5 and 2.5, and 0 elsewhere. If the value falls outside the region [-2.5, 2.5], weight matrices will not be updated.

Figure 4.14 shows simulation results with piecewise linear activation functions. As shown in this figure, all the sigmoid approximation functions show similar results with the actual sigmoid function. This suggests that combining the use of piecewise approximation in the forward path and non-piecewise version of gradients in the backword path can reduce computation.

Since stochastic rounding showed comparable results with floating point, stochastic

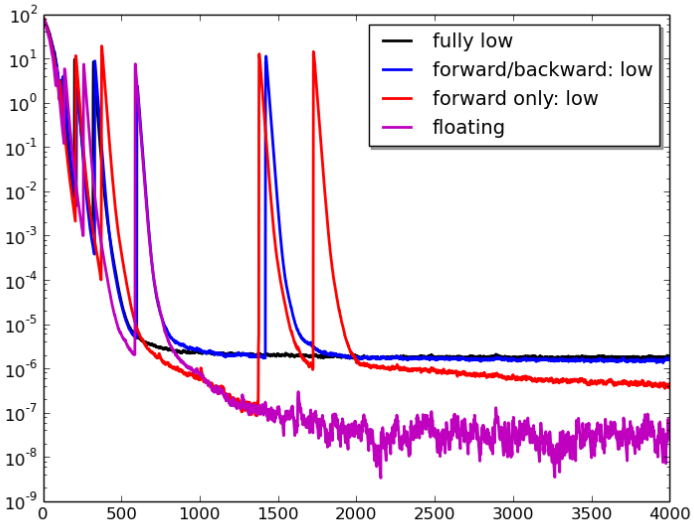


Figure 4.13: Loss with fully low precision, forward/backward low precision, forward only low precision and floating point as a reference. < bit-width: 24, round to nearest, overflow rate: 0.005>

rounding is also applied together with piecewise linear activation function. Figure 4.15 shows simulation results. Together with stochastic rounding, ultra_fast_sigmoid shows similar results with actual sigmoid. Also, hard_sigmoid shows almost similar results with sigmoid. One can implement effective low precision training with piecewise linear activation function when used with stochastic rounding.

4.5 Hardware Implementation

Based on the simulation results, 24-bit fixed point with stochastic rounding shows comparable results with floating point precision. Therefore, low precision hardware is implemented and compared with floating point MAC for energy and performance analysis. Figure 4.16 shows the proposed hardware block diagram. Adding uniform random number between 0 and the smallest positive number ($2^{-\mathtt{FL}3}$) is realized with LFSR filter and mask function. mask bits are thermometer code indicating cropping point (0 to 1 transition on mask bits). Cropping point is determined by adding two `FL`s of the inputs and subtracting `FL` of the

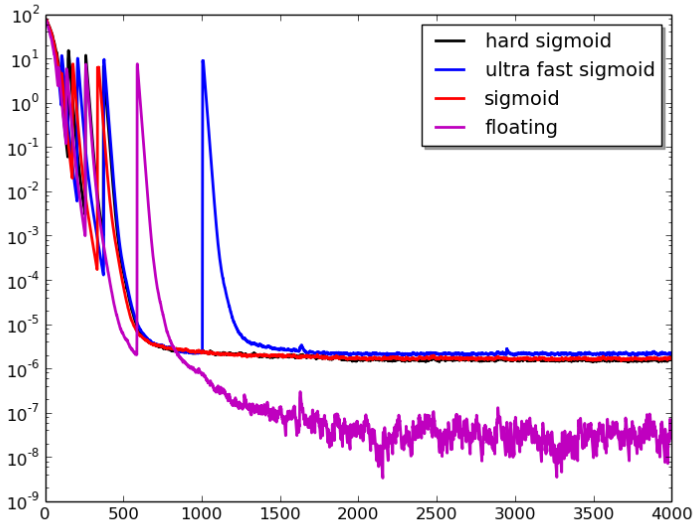


Figure 4.14: Loss with hard_sigmoid, ultra_fast_sigmoid and sigmoid. < bit-width: 24, round to nearest, overflow rate: 0.005, and fully low precision training>

Table 4.1: MAC Unit Implementation Summary

	24-bit fixed point w/ stochastic rounding	32-bit floating point
Area (um^2)	13,299	24,125
Clock (ns)	1.6	4
Power (mW)	4.01	2.623

output. The final output is cropped with 24-bit at the cropping point.

The proposed 24-bit dynamic fixed point MAC with a stochastic rounding unit is implemented with Synopsys 28nm PDK. 32-bit floating point MAC is also implemented for comparison. Table 4.1 summarizes implementation results. The proposed hardware has small foot print and runs at a higher frequency than 32-bit floating point MAC. The proposed hardware is implemented on systolic array [79, 78] to analyze entire system performance. Fig. 17 shows implementation results. The proposed hardware is smaller than 32-bit floating point MAC, thus, number of MAC units integrated in the systolic array is larger than the systolic array with 32-bit floating point MAC unit.

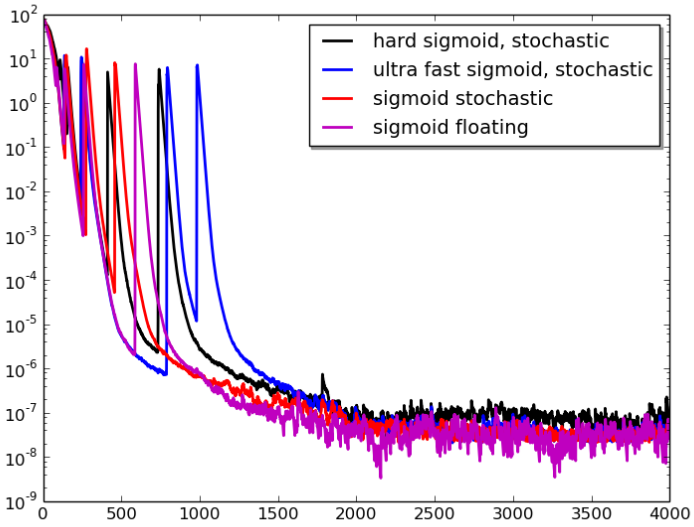


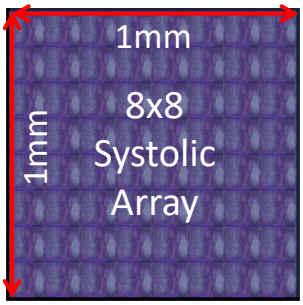
Figure 4.15: Loss with hard_sigmoid, ultra_fast_sigmoid and sigmoid. < bit-width: 24, stochastic rounding, overflow rate: 0.005, and fully low precision training>

As a result, the total one mini-batch training time of the proposed hardware is 4.7x faster than that of 32-bit floating point hardware. Also, the proposed hardware requires 3.09x lower energy than 32-bit floating point hardware.

Since stochastic rounding operation is not needed most of the time (it takes only 2.2% of the entire operations), clock gating can be applied for stochastic rounding to reduce unnecessary toggling. Dynamic power consumption for the stochastic rounding unit is 1.36mW (34% of the entire power consumption), thus, resulting energy becomes 4.55x lower than that of floating point MAC array.

4.6 Summary

This section studies low precision training of RNNs, especially for recently proposed GRU. Dynamic fixed point format is used as a target numeric format and explore low precision training extensively. Manual BPTT calculation for GRU is implemented to emulate low precision training. The study shows batch normalization is essential to enable successful low precision training. The research shows even 64-bit fixed point is not sufficient to



(a)

Systolic Array (1mm^2)		24-bit fixed point w/ stochastic rounding	32-bit floating point
row size		8	6
column size		8	6
# cycles/forward		24K	44K
# cycles/backward		52K	100K
1 step mini-batch training time (ms)		24.3 (**4.7x faster**)	114.7
Energy/step (μJ)	w/o clock gating	97.4 (**3.09x efficient**)	301
	w/ clock gating for rounding	66.2 (**4.55x efficient**)	

(b)

Figure 4.16: (a) 28nm implementation layout of systolic array of 24-bit MAC with stochastic rounding, and (b) comparison with 32-bit floating point counterpart.

train RNNs, and the overflow rate threshold that determines decimal point search has to be kept small. Stochastic rounding with 24-bit precision shows comparable results with floating point precision. Applying stochastic rounding in gradients accumulation during the backward pass is more effective than the any other passes. Piecewise linear activation gives almost similar results with actual nonlinear function which enables further speed-up by removing exponential terms. Systolic array of 24-bit dynamic fixed point MAC with stochastic rounding unit is implemented with Synopsys 28nm PDK for energy and performance analysis. Implementation results show the proposed hardware achieves 4.7x faster training and 4.55x lower energy than 32-bit floating point MAC array.

CHAPTER 5
SECURE DEEP LEARNING

5.1 Introduction

Deep neural networks and other machine learning classifiers are shown to be vulnerable to small perturbations to inputs [50, 51, 23, 52, 24]. Previous works [23, 24, 25] have shown that injecting adversarial examples during training (adversarial training) increases the robustness of a network against adversarial attacks. The adversarial examples can be generated by perturbing the inputs in one step or iteratively to either minimize confidence on true labels or increase confidence of a target false label (section 5.2.1). The networks trained with one-step methods have shown noticeable robustness against one-step attacks, but, limited robustness against iterative attacks at test time. In [24], authors explained that using iterative methods for training didn't help improve robustness against iterative attacks at test time. To address this challenge, The research has made the following contributions:

Cascade adversarial training: The research first shows that iteratively generated adversarial images transfer well between networks when the source and the target networks are trained with the *same training method*. Inspired by this observation, the research proposes *cascade adversarial training* which transfers the knowledge of the end results of adversarial training. In particular, the proposed approach trains a network by injecting iter_FGSM images (section 5.2.1) crafted from an already *defended* network (a network trained with adversarial training) in addition to the one-step adversarial images crafted from the network being trained. The concept of using already trained networks for adversarial training is also introduced in [80]. In their work, purely trained networks are used as another source networks to generate one-step adversarial examples for training. On the contrary, the cascade adversarial training uses already *defended* network for iter_FGSM

images generation.

Low level similarity learning: The study advances the previous data augmentation approach [24] by adding additional regularization in deep features to encourage a network to be insensitive to adversarial perturbation. In particular, the proposed approach injects adversarial images in the mini batch *without replacing* their corresponding clean images and penalize distance between embeddings from the clean and the adversarial examples. There are past examples of using embedding space for learning similarity of high level features like face similarity between two different images [81, 82, 83]. Instead, the embedding space is used for learning similarity of the pixel level differences between two similar images. The intuition of using this regularization is that *small difference on input should not drastically change the high level feature representation*.

Analysis of adversarial training: To show the effectiveness of the algorithms, ResNet models [3] are used on MNIST [84] and CIFAR10 dataset [85] using the proposed adversarial training. The study first shows low level similarity learning improves robustness of the network against adversarial images generated by *one-step* and *iterative* methods compared to the prior work. Modifying the weight of the distance measure in the loss function can help control trade-off between accuracies for the clean and adversarial examples. Together with cascade adversarial training and low-level similarity learning, accuracy increase against unknown iterative attacks is achieved, but at the expense of decreased accuracy for one-step attacks. Finally, the study also shows the proposed *cascade adversarial training and low level similarity learning* provide much better robustness against black box attack. Code to reproduce the experiments is available at `https://github.com/taesikna/cascade_adv_training`.

5.2 Background on Adversarial Attacks

5.2.1 Attack Methods

One-step fast gradient sign method (FGSM), referred to as "step_FGSM", generates adversarial image $\boldsymbol{X}^{adv}$ by adding sign of the gradients w.r.t. the clean image $\boldsymbol{X}$ multiplied by $\epsilon \in [0, 255]$ as shown below [23]:

$$\boldsymbol{X}^{adv} = \boldsymbol{X} + \epsilon \ \text{sign}(\nabla_X J(\boldsymbol{X}, y_{true}))$$

One-step target class method generates $\boldsymbol{X}^{adv}$ by subtracting sign of the gradients computed on a target false label as follows:

$$\boldsymbol{X}^{adv} = \boldsymbol{X} - \epsilon \ \text{sign}(\nabla_X J(\boldsymbol{X}, y_{target}))$$

Least likely class y_{LL} is used as a target class and refer this method as "step_ll".

Basic iterative method, referred to as "iter_FGSM", applies FGSM with small α multiple times.

$$\boldsymbol{X}_0^{adv} = \boldsymbol{X}, \quad \boldsymbol{X}_N^{adv} = Clip_{X,\epsilon}\big\{\boldsymbol{X}_{N-1}^{adv} + \alpha \ \text{sign}(\nabla_{X_{N-1}^{adv}} J(\boldsymbol{X}_{N-1}^{adv}, y_{true}))\big\}$$

α = 1, number of iterations N to be $\min(\epsilon + 4, 1.25\epsilon)$ is used in this study. $Clip_{X,\epsilon}$ is elementwise clipping function where the input is clipped to the range $[\max(0, X - \epsilon), \min(255, X + \epsilon)]$.

Iterative least-likely class method, referred to as "iter_ll", is to apply "step_ll" with small α multiple times.

$$\boldsymbol{X}_0^{adv} = \boldsymbol{X}, \quad \boldsymbol{X}_N^{adv} = Clip_{X,\epsilon}\big\{\boldsymbol{X}_{N-1}^{adv} - \alpha \ \text{sign}(\nabla_{X_{N-1}^{adv}} J(\boldsymbol{X}_{N-1}^{adv}, y_{LL}))\big\}$$

Table 5.1: CIFAR10 test results (%) under black box attacks for ϵ=16. Source networks share the same initialization which is different from the target networks. {Target: R20, R20$_{\boldsymbol{K}}$: standard, ***K***urakin's, Source: R20$_{\mathbf{2}}$, R20$_{\boldsymbol{K2}}$: standard, ***K***urakin's.}

Target	Source: step_FGSM		Source: iter_FGSM	
	R20$_{\mathbf{2}}$	R20$_{\boldsymbol{K2}}$	R20$_{\mathbf{2}}$	R20$_{\boldsymbol{K2}}$
R20	**16.2**	31.6	**2.7**	60.1
R20$_{\boldsymbol{K}}$	**66.7**	82.7	55.8	**28.5**

Carlini and Wagner attack [86] referred to as "CW" solves an optimization problem which minimizes both an objective function f (such that attack is success if and only if $f(\boldsymbol{X}^{adv}) < 0$) and a distance measure between $\boldsymbol{X}^{adv}$ and $\boldsymbol{X}$.

Black box attack is performed by testing accuracy on a target network with the adversarial images crafted from a source network different from the target network. Lower accuracy means successful black-box attack. This attack is referred to as white-box attack.

5.2.2 Defense Methods

Adversarial training [24]: is a form of data augmentation where it injects adversarial examples during training. In this method, k examples are taken from the mini batch B (size of m) and the adversarial examples are generated with one of step method. The k adversarial examples *replaces* the corresponding clean examples when making mini batch. Below this adversarial training method is referred to as "Kurakin's".

Ensemble adversarial training [80]: is essentially the same with the adversarial training, but uses several pre-trained vanilla networks to generate one-step adversarial examples for training. Below this adversarial training method referred to as "Ensemble".

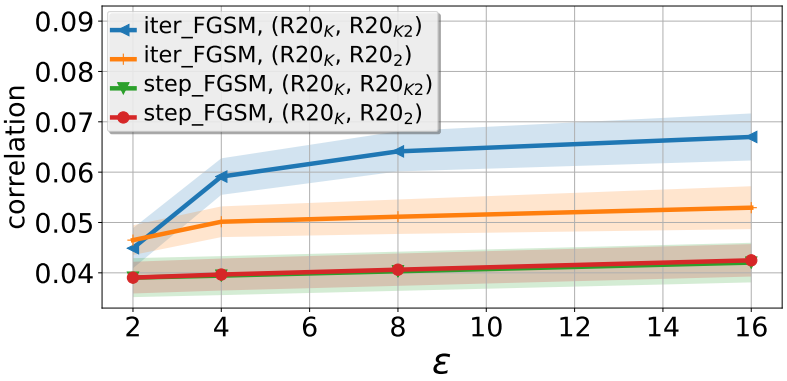


Figure 5.1: Correlation between adversarial noises from different networks for each ϵ. Shaded region shows $\pm$ 0.1 standard deviation of each line.

5.3 Proposed Approach

5.3.1 Transferability Analysis

In this section, transferability between purely trained networks and adversarially trained networks is shown under black box attack. ResNet [3] models are used for CIFAR10 classification. 20-layer ResNets are trained with different methods (standard training, adversarial training [24]) and those are used as target networks. Networks (standard training and adversarial training) are retrained with the different initialization from the target networks, and the trained networks are used as source networks. Experimental details can be found in Section 5.4. In Table 5.1, test accuracies under black box attack are reported.

Transferability (step attack): First, high robustness against one-step attack between defended networks (R20$_{\boldsymbol{K2}}$ -> R20$_{\boldsymbol{K}}$) and low robustness between undefended networks (R20$_{\mathbf{2}}$ -> R20) are observed. This observation shows that error surfaces of neural networks are *driven by the training method* and networks trained with the same method end up similar optimum states.

It is noteworthy to observe that the accuracies against step attack from the undefended network (R20$_{\mathbf{2}}$) are always lower than those from defended network (R20$_{\boldsymbol{K2}}$). Possible explanation for this would be that adversarial training tweaks gradient seen from the clean

image to point toward weaker adversarial point along that gradient direction. As a result, one-step adversarial images from defended networks become weaker than those from undefended network.

Transferability (iterative attack): "iter_FGSM" attack remains very strong even under the black box attack scenario but *only between undefended networks or defended networks*. This is because iter_FGSM noises (X^{adv}-X) from defended networks resemble each other. As shown in Figure 5.1, higher correlation between iter_FGSM noises from a defended network (R20$_{\boldsymbol{K}}$) and those from another defended network (R20$_{\boldsymbol{K2}}$) is observed.

Difficulty of defense/attack under the black box attack scenario: As seen from this observation, it is efficient to attack an undefended/defended network with iter_FGSM examples crafted from another undefended/defended network. Thus, when building a robust network under the black box attack scenario, it is desired to check accuracies for the adversarial examples crafted from other networks trained with the *same strategy*.

5.3.2 Cascade Adversarial Training

Inspired by the observation that iter_FGSM images transfer well between defended networks, *cascade adversarial training* is proposed. The algorithm trains a network by injecting iter_FGSM images crafted from an already defended network. The hypothesis is that the network being trained with cascade adversarial training will learn to avoid such adversarial perturbation, enhancing robustness against iter_FGSM attack. The intuition behind this proposed method is that cascade adversarial training *transfers the knowledge of the end results of adversarial training*. In particular, it trains a network by injecting iter_FGSM images crafted from already *defended* network in addition to the one-step adversarial images crafted from the network being trained.

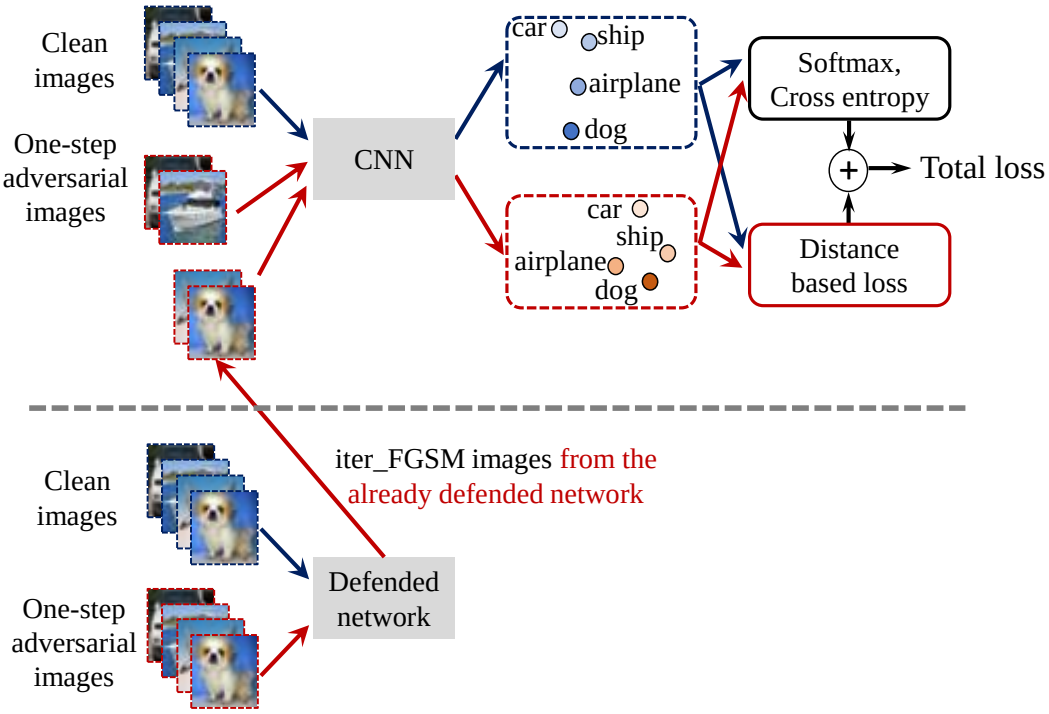


Figure 5.2: Cascade adversarial training regularized with a unified embedding.

5.3.3 Regularization with a Unified Embedding

The algorithm proposed in [24] is advanced by adding low level similarity learning. Unlike [24], the clean examples used for generating adversarial images are also included in the mini batch. Once one step forward pass is performed with the mini batch, embeddings are followed by the softmax layer for the cross entropy loss for the standard classification. At the same time, clean embeddings and adversarial embeddings are taken, and the distance between the two is minimized with the distance based loss.

The distance based loss encourages two similar images (clean and adversarial) to produce the same outputs, *not necessarily the true labels*. Thus, low-level similarity learning can be considered as an unsupervised learning. By adding regularization in higher embedding layer, convolution filters *gradually learn* how to ignore such pixel-level perturbation. Tegularization have been applied on lower layers with an assumption that low level pixel

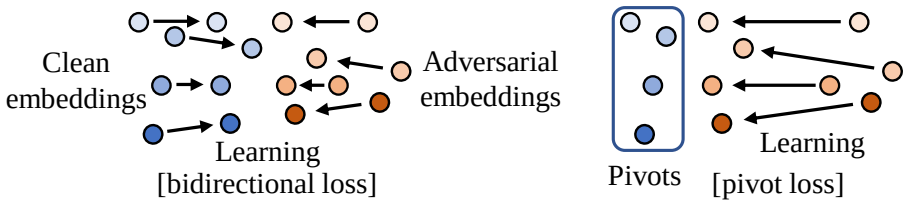


Figure 5.3: (**Left**) Bidirectional loss. (**Right**) Pivot loss.

perturbation can be ignored in lower hierarchy of networks. However, adding regularization term on higher embedding layer right before the softmax layer showed best performance. *The more convolutional filters have chance to learn such similarity, the better the performance.* Note that cross entropy doesn't encourage two similar images to produce the same output labels. Standard image classification using cross entropy compares ground truth labels with outputs of a network *regardless of* how similar training images are.

The entire training process combining cascade adversarial training and low level similarity learning is shown in Figure 5.2. The total loss is defined as follows:

$$Loss = \frac{1}{(m-k)+\lambda k}\left(\sum_{i=1}^{m-k} L(\boldsymbol{X}_i|y_i) + \lambda \sum_{i=1}^{k} L(\boldsymbol{X}_i^{adv}|y_i)\right) + \lambda_2 \sum_{i=1}^{k} L_{dist}(\boldsymbol{E}_i^{adv}, \boldsymbol{E}_i)$$

$\boldsymbol{E}_i$ and $\boldsymbol{E}_i^{adv}$ are the resulting embeddings from $\boldsymbol{X}_i$ and $\boldsymbol{X}_i^{adv}$, respectively. m is the size of the mini batch, k ($\leq m/2$) is the number of adversarial images in the mini batch. λ is the parameter to control the relative weight of classification loss for adversarial images. λ_2 is the parameter to control the relative weight of the distance based loss L_{dist} in the total loss.

Bidirectional loss minimizes the distance between the two embeddings by moving both clean and adversarial embeddings as shown in the left side of the Figure 5.3.

$$L_{dist}(\boldsymbol{E}_i^{adv}, \boldsymbol{E}_i) = ||\boldsymbol{E}_i^{adv} - \boldsymbol{E}_i||_N^N, \quad N \in 1, 2 \quad i = 1, 2, ..., k$$

N = 1 and 2 were tried and there was not much difference between the two. The results with N = 2 are reported for the rest of the chapter otherwise noted. When N = 2, L_{dist}

becomes L2 loss.

Pivot loss minimizes the distance between the two embeddings by moving only the adversarial embeddings as shown in the right side of the Figure 5.3.

$$L_{dist}(\boldsymbol{E}_i^{adv}|\boldsymbol{E}_i) = ||\boldsymbol{E}_i^{adv} - \boldsymbol{E}_i||_N^N, \quad N \in 1,2 \quad i = 1,2,...,k$$

In this case, clean embeddings ($\boldsymbol{E}_i$) serve as pivots to the adversarial embeddings. [1] In particular, back-propagation through the clean embeddings is not performed for the distance based loss. The intuition behind the use of pivot loss is that the embedding from a clean image can be treated as the ground truth embedding.

5.4 Experimental Setup

This section describes experimental setup. The image values are scaled down to [0,1] and no data augmentation is performed for MNIST. For CIFAR10, the image values are scaled down to [0,1] and subtracted by per-pixel mean values. 24x24 random crop and random flip are performed on 32x32 original images. [2] Adversarial images are generated with "step_ll" after these steps unless otherwise noted.

For adversarial training, k = 64 adversarial examples are generated among 128 images in one mini-batch. As in [24], randomly chosen ϵ is used in the interval [0, max_e] with clipped normal distribution $N(\mu = 0, \sigma = max\_e/2)$, where max_e is the maximum ϵ used in training. max_e = 0.3*255 is used for MNIST, and 16 for CIFAR10. λ = 0.3 and λ_2 = 0.0001 are used in the loss function.

Stochastic gradient descent (SGD) optimizer is used with momentum of 0.9. Weight decay is 0.0001. Training is started with a learning rate of 0.1. The learning rate is then divided by 10 at 4k and 6k iterations for MNIST, and 48k and 72k iterations for CIFAR10.

[1] In Tensorflow, non-trainable variable is created and this is updated with the clean embedding E_i at every training step.

[2] Original paper [3] performed 32x32 random crop on zero-padded 40x40 images.

Table 5.2: MNIST test results (%) for 20-layer ResNet models (ϵ = 0.3*255 at test time). { R20M: standard training, R20M$_{\boldsymbol{K}}$: $\boldsymbol{K}$urakin's adversarial training, R20M$_{\boldsymbol{B}}$: $\boldsymbol{B}$idirectional loss, R20M$_{\boldsymbol{P}}$: $\boldsymbol{P}$ivot loss.} CW L_∞ attack is performed with 100 test samples (10 samples per each class) and the number of adversarial examples with $\epsilon > 0.3$*255 is reported. Additional details for CW attack can be found in Appendix A.3

Model	clean	step_ll	step_FGSM	iter_ll	iter_FGSM	CW
R20M	99.6	9.7	10.3	0.0	0.0	0
R20M$_{\boldsymbol{K}}$	99.6	96.7	94.5	89.0	60.2	46
R20M$_{\boldsymbol{B}}$ (Proposed)	99.5	**97.3**	**96.2**	**97.2**	**88.5**	**81**
R20M$_{\boldsymbol{P}}$ (Proposed)	99.5	**97.1**	**95.7**	**96.9**	**88.9**	**82**

Training is terminated at 8k iterations for MNIST, and 94k iterations for CIFAR10. [3]

Initialization affects the training results slightly as in [24], thus, the pre-trained networks are used as initial starting points for different configurations. Pre-training is done with 2 and 10 epochs for MNIST and CIFAR10.

5.5 Low Level Similarity Learning Analysis

5.5.1 Experimental Results on MNIST

In this section, the effect of low level similarity learning is first analyzed on MNIST. ResNet models [3] are trained with different methods (standard training, Kurakin's adversarial training and adversarial training with the distance based loss). Experimental details can be found in Section 5.4.

Table 5.2 shows the accuracy results for MNIST test dataset for different types of attack methods. As shown in the table, the proposed method achieves better accuracy than Kurakin's method for all types of attacks with a little sacrifice on the accuracy for the clean images. Even though adversarial training is done only with "step_ll", additional regularization increases robustness against *unknown* "step_FGSM", "iter_ll", "iter_FGSM" and CW L_∞ attacks. This shows that the low-level similarity learning can successfully regularize

[3]The adversarial training requires longer training time than the standard training. Authors in the original paper [3] changed the learning rate at 32k and 48k iterations and terminated training at 64k iterations.

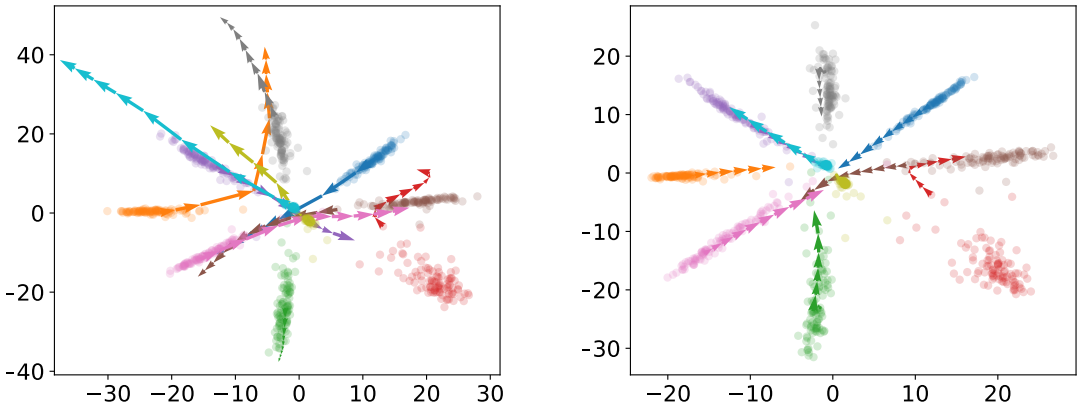
the one-step adversarial perturbation and its vicinity for simple image classification like MNIST.

5.5.2 Embedding Space Visualization

To visualize the embedding space, 20-layer ResNet model is modified. The last fully connected layer (64x10) is changed to two fully connected layers (64x2 and 2x10). Networks are re-trained with standard training, Kurakin's method and with the pivot loss on MNIST. [4]

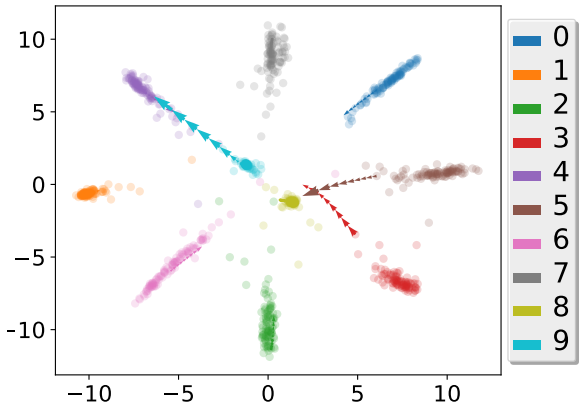
In Figure 5.4, embeddings (dimension=2) between two fully connected layers are drawn. As seen from this figure, adversarial images from the network trained with standard training cross the decision boundary easily as ϵ increases. With Kurakin's adversarial training, the distances between clean and adversarial embeddings are minimized compared to standard training. And the pivot loss further minimizes distance between the clean and adversarial embeddings. Note that the pivot loss also decreases absolute value of the embeddings, thus, higher λ_2 will eventually result in overlap between distinct embedding distributions. Intra class variation of the clean embeddings is also minimized for the network trained with the pivot loss as shown in the scatter plot in Figure 5.4 (c).

[4] Modified ResNet models showed slight decreased accuracy for both clean and adversarial images compared to original ResNet counterparts, however, similar trend is observed (improved accuracy for iterative attacks for the network trained with pivot loss) as in Table 5.2.



(a) Standard

(b) ***K***urakin



(c) ***P***ivot (Proposed)

Figure 5.4: Embedding space visualization for modified ResNet models trained on MNIST. x-axis and y-axis show first and second dimension of embeddings respectively. Scatter plot shows first 100 clean embeddings per each class on MNIST test set. Each arrow shows difference between two embeddings (one from iter_FGSM image (ϵ) and the other from (ϵ+8)). Arrows are drawn from $\epsilon = 0$ to $\epsilon = 76$ ($\approx$ 0.3*255) for one sample image per each class. As shown in this figure, differences between clean and corresponding adversarial embeddings are minimized for the network trained with pivot loss.

5.5.3 Experimental Results on CIFAR10

Table 5.3 shows the accuracy results for 20-layer ResNet models on CIFAR10 test dataset. Again, the proposed low-level similarity learning further improves robustness against all types of attacks compared to Kurakin's adversarial training. However, the accuracy improvements against iterative attacks (iter_FGSM, CW) are limited, showing regularization

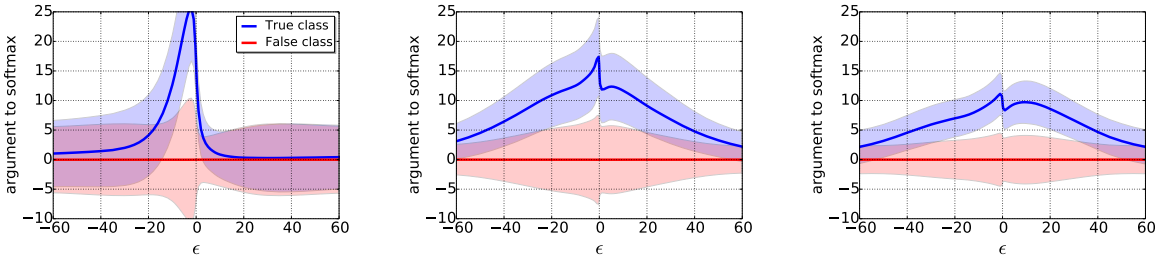
Table 5.3: CIFAR10 test results (%) for 20-layer ResNet models. { R20: standard training, R20$_{\boldsymbol{K}}$: $\boldsymbol{K}$urakin's adversarial training, R20$_{\boldsymbol{B}}$: $\boldsymbol{B}$idirectional loss, R20$_{\boldsymbol{P}}$: $\boldsymbol{P}$ivot loss.} CW L_∞ attack is performed with 100 test samples (10 samples per each class) and the number of adversarial examples with $\epsilon > 2$ or 4 is reported.

Model	clean	step_ll		step_FGSM		iter_FGSM		CW	
		ϵ=2	ϵ=16	ϵ=2	ϵ=16	ϵ=2	ϵ=4	ϵ=2	ϵ=4
R20	90.9	44.5	11.5	28.7	12.2	14.0	0.4	8	0
R20$_{\boldsymbol{K}}$	91.0	85.8	84.4	78.9	81.5	50.6	9.8	13	2
R20$_{\boldsymbol{B}}$ (Proposed)	91.0	86.1	**87.3**	78.7	**90.0**	**52.0**	**11.2**	**19**	**3**
R20$_{\boldsymbol{P}}$ (Proposed)	90.9	86.2	**88.3**	79.7	**91.7**	**52.0**	**11.3**	**18**	**4**

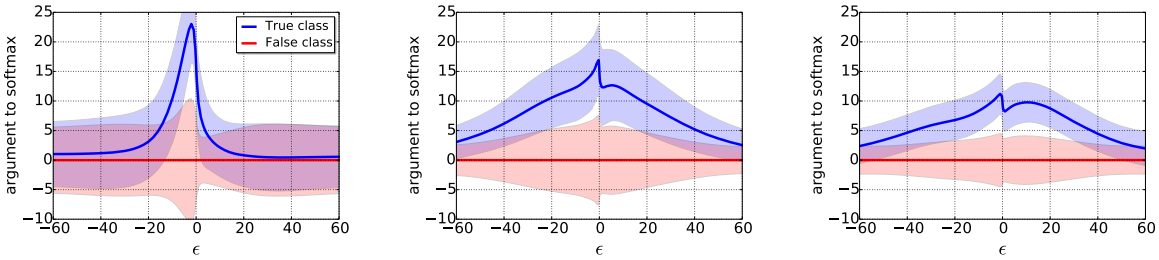
effect of low-level similarity learning is not sufficient for the iterative attacks on complex color images like CIFAR10.

5.5.4 Alternative Visualization on Embeddings

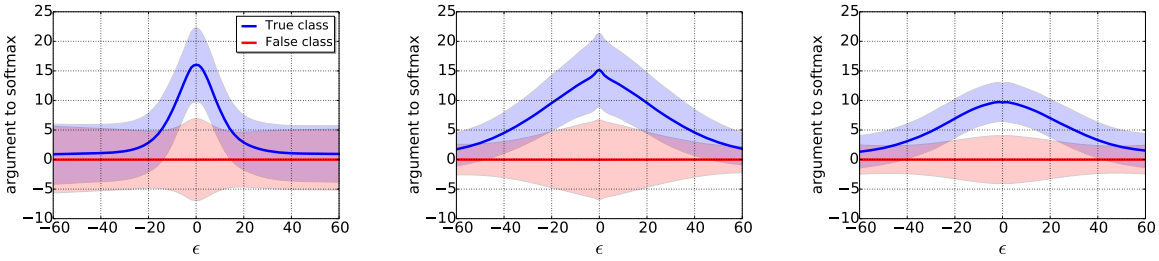
In this section, average value of the argument to the softmax layer is drawn for the true class and the false classes to visualize how the adversarial training works as in Figure 5.5. Standard training, as expected, shows dramatic drop in the values for the true class as ϵ is increased in "step_ll" or "step_FGSM direction. With adversarial training, the value drop is limited at small ϵ and the proposed method even increases the value in certain range upto ϵ=10. Note that adversarial training is not the same as the gradient masking. As illustrated in Figure 5.5, it exposes gradient information, however, quickly distort gradients along the sign of the gradient ("step_ll" or "step_FGSM) direction.



(a) R20, step_ll (b) R20$_{\boldsymbol{K}}$, step_ll (c) R20$_{\boldsymbol{P}}$, step_ll



(d) R20, step_FGSM (e) R20$_{\boldsymbol{K}}$, step_FGSM (f) R20$_{\boldsymbol{P}}$, step_FGSM



(g) R20, random sign (h) R20$_{\boldsymbol{K}}$, random sign (i) R20$_{\boldsymbol{P}}$, random sign

Figure 5.5: Argument to the softmax vs. ϵ in test time. "step_ll", "step_FGSM" and "random sign" methods were used to generate test-time adversarial images. Arguments to the softmax were measured by changing ϵ for each test method and averaged over randomly chosen 128 images from CIFAR10 test-set. Blue line represents true class and the red line represents mean of the false classes. Shaded region shows $\pm$ 1 standard deviation of each line.

Improved results (broader margins than baseline) are observed for "random sign" added images even though the random sign added images are not used during training. Overall shape of the argument to the softmax layer in the proposed case becomes smoother than Kurakin's method, suggesting the proposed method is good for pixel level regularization. Even though actual value of the embeddings for the true class in R20$_{\boldsymbol{P}}$ is smaller than that in R20$_{\boldsymbol{K}}$, the standard deviation of R20$_{\boldsymbol{P}}$ is less than R20$_{\boldsymbol{P}}$, making better margin between

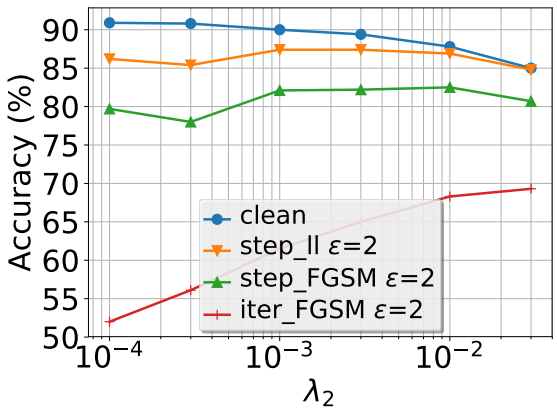


Figure 5.6: Accuracy vs. λ_2

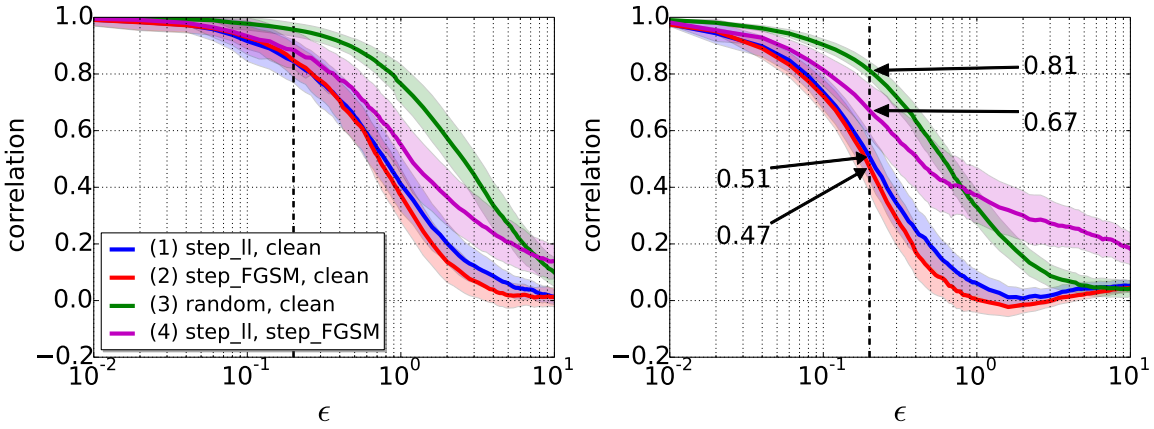
the true class and false classes.

5.5.5 Effect of λ_2 on CIFAR10

In this section, 20-layer ResNet models are trained with various λ_2s for CIFAR10 dataset to study effects of the weight of the distance measure in the loss function. Figure 5.6 shows that a higher λ_2 increases accuracy of the iteratively generated adversarial images. However, it reduces accuracy on the clean images, and increasing λ_2 above 0.3 even results in divergence of the training. This is because embedding distributions of different classes will eventually overlap since absolute value of the embedding will be decreased as λ_2 increases as seen from the section 5.5.2. This experiment results show that there exists clear trade-off between accuracy for the clean images and that for the adversarial images. Thus, it is recommend using a very high λ_2 only under strong adversarial environment.

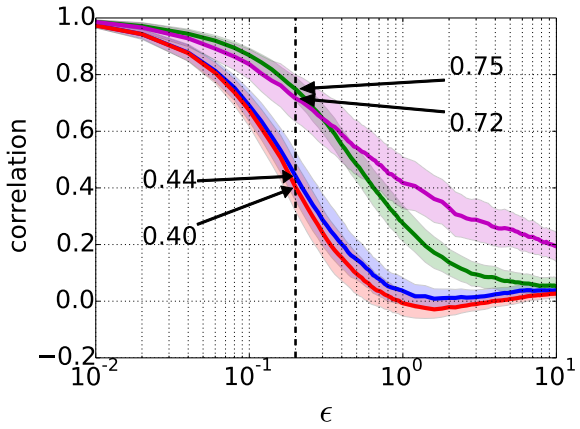
5.6 Label Leaking Analysis

Accuracies for the "step_FGSM" adversarial images become higher than those for the clean images ("label leaking" phenomenon) by training with "step_FGSM" examples as in [24]. Interestingly, "label leaking" phenomenon is also observed even without providing true labels for adversarial images generation. "Label leaking" is a natural result of the adversarial



(a) R20: Baseline

(b) R20$_K$: Kurakin's



(c) R20$_P$: Pivot (Proposed)

Figure 5.7: Averaged Pearson's correlation coefficient between the gradients w.r.t. two images. Correlation were measured by changing ϵ for each adversarial image and averaged over randomly chosen 128 images from CIFAR10 test-set. Shaded region represents $\pm$ 0.5 standard deviation of each line.

training.

To understand the nature of adversarial training, correlation between gradients w.r.t. different images (i.e. clean vs. adversarial) is measured as a measure of error surface similarity. In particular, correlation is measured between gradients w.r.t. (1) clean vs. "step_ll" image, (2) clean vs. "step_FGSM" image, (3) clean vs. "random sign" added image, and (4) "step_ll" image vs. "step_FGSM" image for three trained networks (a) R20, (b) R20$_K$ and (c) R20$_P$ (Proposed) in Table 5.3. Figure 5.7 draws average value of

correlations for each case.

Meaning of the correlation: In order to make strong adversarial images with "step_FGSM" method, correlation between the gradient w.r.t. the clean image and the gradient w.r.t. its corresponding adversarial image should remain high since the "step_FGSM" method *only* use the gradient w.r.t. the clean image (ϵ=0). Lower correlation means perturbing the adversarial image at ϵ further to the gradient (seen from the clean image) direction is no longer efficient.

Results of adversarial training: (1) and (2) become quickly lower than (3) as ϵ increases. This means that, when the inputs are provided toward the steepest (gradient) direction on the error surface, gradient is more quickly uncorrelated with the gradient w.r.t. the clean image than when the inputs are provided along the random direction. As a result of adversarial training, this uncorrelation is observed at a lower ϵ making one-step attack less efficient even with small perturbation. (1), (2) and (3) for R20$_{\boldsymbol{P}}$ are slightly lower than Kurakin's method at the same ϵ which means that the proposed similarity learning is better at defending one-step attacks than Kurakin's.

Error surface similarity between "step_ll" and "step_FGSM" images: (4) remains high with higher ϵ for all trained networks. This means that the error surface (gradient) of the "step_ll" image and that of its corresponding "step_FGSM" image resemble each other. That is the reason why the robustness against "step_FGSM" method is observed only by training with "step_ll" method and vice versa. (4) for R20$_{\boldsymbol{P}}$ is slightly higher than R20$_{\boldsymbol{K}}$ at the same ϵ and that means the similarity learning tends to make error surfaces of the adversarial images with "step_ll" and "step_FGSM" method to be more similar.

Analysis of label leaking phenomenon: Interestingly, (2) becomes slightly negative in certain range ($1 < \epsilon < 3$ for Kurakin's, and $1 < \epsilon < 4$ for Pivot (Proposed)) and this could be the possible reason for "label leaking" phenomenon. For example, let's assume that there is a perturbed image (by "step_FGSM" method) at ϵ where the correlation between the gradients w.r.t. that image and the corresponding clean image is negative. Further

Table 5.4: CIFAR10 test results (%) for 20 trained networks (Kurakin's and the proposed method w/ pivot loss).

			step_ll attack		iter_ll attack	
ϵ		0	4	16	4	16
mean $\pm$	Kurakin's	90.9 ± 0.25	86.1 ± 0.98	84.6 ± 2.89	64.2 ± 2.89	16.3 ± 2.02
std	Pivot	90.9 ± 0.25	$87.3 \pm \mathbf{0.79}$	$87.8 \pm \mathbf{0.82}$	$69.4 \pm \mathbf{1.61}$	$21.9 \pm \mathbf{1.85}$

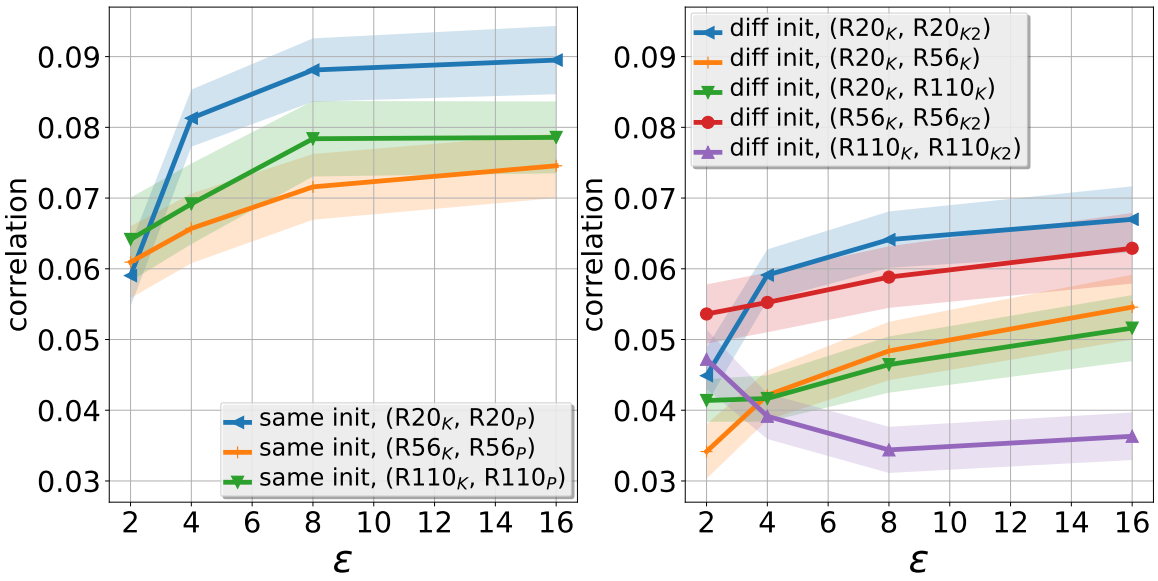
increase of ϵ with the gradient (w.r.t. the clean image) direction actually decreases the loss resulting in increased accuracy (label leaking phenomenon). Due to the error surface similarity between "step_ll" and "step_FGSM" images and this negative correlation effect, however, label leaking phenomenon can always happen for the networks trained with *one-step* adversarial examples.

5.6.1 Effect of Variations in Initial Condition

To see how the training is sensitive to the initialization, 20 networks are trained [5] with Kurakin's method and the proposed method with pivot loss ("step_ll"). Mean accuracy and the standard deviation are reported.

As shown in Table 5.4, the proposed method produces higher mean accuracies and lower standard deviation of accuracy than Kurakin's method. This suggests that the low-level similarity learning also reduces uncertainties in training, thus, makes networks to be converged similarly. Interestingly, *networks robust to one-step attacks tend to be less robust to iterative attacks and vice versa* for both methods, however, this variation was less in the proposed case than in Kurakin's case. Difficulty of optimizing networks for both one-step attacks and iterative attacks can also be found in the following section.

[5] Adversarial training without distance based loss sometimes ended up with poor results. Thus, a learning rate was scheduled at 60k and 90k iterations, and the training is terminated at 120k iterations for this experiment.



(a) Corr: same initialization

(b) Corr: different initialization

Figure 5.8: Correlation between iter_FGSM noises crafted from different networks for each ϵ. Correlation is averaged over randomly chosen 128 images from CIFAR10 test-set.

5.7 Cascade Adversarial Training Analysis

5.7.1 Source Network Selection

In this section, the transferability of iter_FGSM images between various architectures is provided. To this end, 56-layer ResNet networks (Kurakin's, pivot loss) are trained with the same initialization. Then another 56-layer ResNet network (Kurakin's) are trained with different initialization. Training for the 110-layer ResNet networks are repeated. Correlation between iter_FGSM noises are measured from different networks.

Figure 5.8 (a) shows correlation between iter_FGSM noises crafted from Kurakin's network and those from Pivot network with the same initialization. Conjectured from [24], high correlation between iter_FGSM noises is observed from networks with the same initialization. Correlation between iter_FGSM noises from the networks with different initialization, however, becomes lower as the network is deeper as shown in Figure 5.8 (b). Since the degree of freedom increases as the network size increases, adversarially trained

Table 5.5: CIFAR10 test results (%) for 110-layer ResNet models. CW L_∞ attack is performed with 100 test samples (10 samples per each class) and the number of adversarial examples with $\epsilon > 2$ or 4 is reported. {R110$_{\boldsymbol{K}}$: $\boldsymbol{K}$urakin's, R110$_{\boldsymbol{P}}$: $\boldsymbol{P}$ivot loss, R110$_{\boldsymbol{E}}$: $\boldsymbol{E}$nsemble training, R110$_{\boldsymbol{K},\boldsymbol{C}}$: $\boldsymbol{K}$urakin's and $\boldsymbol{C}$ascade training, R110$_{\boldsymbol{P},\boldsymbol{E}}$: $\boldsymbol{P}$ivot loss and $\boldsymbol{E}$nsemble training, and R110$_{\boldsymbol{P},\boldsymbol{C}}$: $\boldsymbol{P}$ivot loss and $\boldsymbol{C}$ascade training}

Model	clean	step_ll		step_FGSM		iter_FGSM		CW	
		ϵ=2	ϵ=16	ϵ=2	ϵ=16	ϵ=2	ϵ=4	ϵ=2	ϵ=4
R110$_{\boldsymbol{K}}$	92.3	**88.3**	**90.7**	**86.0**	**95.2**	59.4	9.2	25	4
R110$_{\boldsymbol{P}}$ (Proposed)	92.3	86.0	89.4	81.6	91.6	64.1	20.9	32	7
R110$_{\boldsymbol{E}}$	92.3	86.3	74.3	84.1	72.9	63.5	21.1	24	6
R110$_{\boldsymbol{K},\boldsymbol{C}}$ (Proposed)	92.3	86.2	72.8	82.6	66.7	69.3	33.4	20	5
R110$_{\boldsymbol{P},\boldsymbol{E}}$ (Proposed)	91.3	84.0	65.7	77.6	54.5	66.8	38.3	**38**	**16**
R110$_{\boldsymbol{P},\boldsymbol{C}}$ (Proposed)	91.5	85.7	76.4	82.4	69.1	**73.5**	**42.5**	27	15

networks prone to end up with different states, thus, making transfer rate lower. To maximize the benefit of the cascade adversarial training, use of *the same initialization for a cascade network and a source network* (used for iterative adversarial examples generation) is desired.

5.7.2 White Box Attack Analysis

A network trained with Kurakin's method and that with pivot loss is first compared. 110-layer ResNet models with/without pivot loss are trained and accuracies are reported in Table 5.5. As shown in the table, the proposed low-level similarity learning further improves robustness against iterative attacks compared to Kurakin's adversarial training. However, the accuracy improvements against iterative attacks (iter_FGSM, CW) are limited, showing regularization effect of low-level similarity learning is not sufficient for the iterative attacks on complex color images like CIFAR10. This is different from MNIST test cases where significant accuracy increase is observed for iterative attacks only with pivot loss. Label leaking phenomenon reported in [24] happens even though the network is not trained with step_FGSM images.

Next, A network is trained from scratch with iter_FGSM examples crafted from the defended network, R110$_{\boldsymbol{P}}$. Initialization used in R110$_{\boldsymbol{P}}$ is also used as discussed in 5.7.1. In particular, iter_FGSM images are crafted from R110$_{\boldsymbol{P}}$ with CIFAR10 training images for ϵ= 1,2, ..., 16, and those are used randomly together with step_ll examples from the network being trained. Cascade networks are trained with/without pivot loss. Networks with ensemble adversarial training [80] are also trained with/without pivot loss for comparison. The implementation details for the trained models can be found in Appendix A.1.

Several meaningful observations are found in Table 5.5. First, ensemble and cascade models show improved accuracy against iterative attack although at the expense of decreased accuracy for one-step attacks compared to the baseline defended network (R110$_{\boldsymbol{K}}$). Additional data augmentation from other networks enhances the robustness against iterative attack, weakening label leaking effect caused by one-step adversarial training.

Second, the proposed low-level similarity learning (R110$_{\boldsymbol{P,E}}$, R110$_{\boldsymbol{P,C}}$) further enhances robustness against iterative attacks including fully unknown CW attack (especially for ϵ=4). Additional knowledge learned from data augmentation through cascade/ensemble adversarial training enables networks to learn partial knowledge of perturbations generated by an iterative method. And the *learned iterative perturbations become regularized further with the low-level similarity learning* making networks robust against unknown iterative attacks.

Third, low-level similarity learning serves as a good regularizer for adversarial images, but not for the clean images for ensemble/cascade models (reduced accuracy for the clean images for R110$_{\boldsymbol{P,E}}$ and R110$_{\boldsymbol{P,C}}$ in Table 5.5). Compared to pure adversarial training with one-step adversarial examples, the network sees more various adversarial perturbations during training as a result of ensemble and cascade training. Those perturbations are prone to end up embeddings in the vicinity of decision boundary more often than perturbations caused by one-step adversarial training. Pivot loss pulls the vicinity of those adversarial embeddings toward their corresponding clean embeddings. During this process,

Table 5.6: CIFAR10 test results (%) for 110-layer ResNet models under black box attacks (ϵ=16). {Target: same networks in Table 5.5 and R110$_{\boldsymbol{K}}$: $\boldsymbol{K}$urakin's, Source: re-trained baseline, Kurakin's, cascade and ensemble networks with/without pivot loss. Source networks use the different initialization from the target networks. Additional details of the models can be found in Appendix A.1.}

Target	Source: iter_FGSM						
	R110$_{\boldsymbol{2}}$	R110$_{\boldsymbol{K2}}$	R110$_{\boldsymbol{E2}}$	R110$_{\boldsymbol{P2}}$	R110$_{\boldsymbol{K,C2}}$	R110$_{\boldsymbol{P,E2}}$	R110$_{\boldsymbol{P,C2}}$
R110$_{\boldsymbol{K}}$	70.5	73.2	**27.9**	77.0	67.3	54.6	80.8
R110$_{\boldsymbol{E}}$	77.9	79.5	55.8	79.0	68.2	**54.7**	82.7
R110$_{\boldsymbol{P}}$ (Proposed)	75.9	75.6	**39.6**	78.5	68.3	61.3	83.3
R110$_{\boldsymbol{K,C}}$ (Proposed)	**56.4**	80.2	61.1	79.5	67.4	62.6	82.1
R110$_{\boldsymbol{P,E}}$ (Proposed)	78.2	82.1	**67.7**	81.7	73.4	68.4	83.8
R110$_{\boldsymbol{P,C}}$ (Proposed)	71.9	80.4	**63.9**	80.1	71.1	64.2	83.0

clean embeddings from other classes might also be moved toward the decision boundary which results in decreased accuracy for the clean images.

5.7.3 Black Box Attack Analysis

In this section, black box attack analysis is presented for the cascade/ensemble networks with/without pivot loss. Black box attack accuracy is reported with the source networks trained with the same method, but with different initialization from the target networks. The reason for this is adversarial examples transfer well between networks trained with the same strategy as observed in section 5.3.1. 110-layer ResNet models are re-trained using Kurakin's, cascade and ensemble adversarial training with/without low-level similarity learning. Those networks are used as source networks for black-box attacks. Baseline 110-layer ResNet model is also included as a source network. Target networks are the same networks used in Table 5.5. Iter_FGSM attack resulted in lower accuracy than step_FGSM attack, thus, iter_FGSM attack results are reported in Table 5.6.

First, iter_FGSM attack from ensemble models (R110$_{\boldsymbol{E2}}$, R110$_{\boldsymbol{P,E2}}$) is strong (results in lower accuracy) compared to that from any other trained networks. [6] Since ensemble

[6]This is also observed when the source and the target networks are switched. Additional details can be

models learn various perturbation during training, adversarial noises crafted from those networks might be more general for other networks making them transfer easily between defended networks.

Second, cascade adversarial training breaks chicken and egg problem. (In section 5.3.1, the analysis shows it is efficient to use a defended network as a source network to attack another defended network.) Even though the transferability between defended networks is reduced for deeper networks, cascade network (R110$_{K,C}$) shows worst case performance against the attack not from a defended network, but from a purely trained network (R110$_{\mathbf{2}}$). Possible solution to further improve the worst case robustness would be to use more than one network as source networks (including pure/defended networks) for iter_FGSM images generation for cascade adversarial training.

Third, ensemble/cascade networks together with the low-level similarity learning (R110$_{P,E}$, R110$_{P,C}$) show better worst case accuracy under black box attack scenario. This shows that enhancing robustness against iterative white box attack also improves robustness against iterative black box attack.

5.8 Summary

This chapter introduced through transfer analysis and showed iter_FGSM images transfer easily between networks trained with the *same strategy*. The research exploited this and proposed cascade adversarial training, a method to train a network with iter_FGSM adversarial images crafted from already defended networks. The study also proposed adversarial training regularized with a unified embedding for classification and low- level similarity learning by penalizing distance between the clean and their corresponding adversarial embeddings. Combining those two techniques (low level similarity learning + cascade adversarial training) with deeper networks further improved robustness against iterative attacks for both white-box and black-box attacks.

found in Appendix A.2.

However, there is still a gap between accuracy for the clean images and that for the adversarial images. Improving robustness against both one-step and iterative attacks still remains challenging since it is shown to be difficult to train networks robust for both one-step and iterative attacks simultaneously. Future research is necessary to further improve the robustness against iterative attack *without* sacrificing the accuracy for step attacks or clean images under both white-box attack and black-box attack scenarios.

CHAPTER 6

NOISE-ROBUST AND RESOLUTION-INVARIANT IMAGE CLASSIFICATION

6.1 Introduction

Image classification using deep learning is being widely adopted for the internet of things (IoTs) [31, 29, 78]. For power hungry edge devices, it is critical to manage the trade-off between energy and quality of the captured image. Region of interest (RoI) based coding is becoming a norm for controlling the energy quality trade-off in resource constraint edge devices [87, 88, 26]. Also, inherent image sensor noise has to be considered for successful image classification for low-end devices [27]. In this chapter, all these image quality degradation including low resolution (LR), random noise and mixed resolution (MR) in a single image due to RoI coding are treated as pixel level perturbation. The objective of this study is to improve the robustness of a classifier against such perturbations.

Many prior work have been studied the impact of low quality images on the image classification. There are two ways to improve the classification accuracy. One is to remove such perturbation itself before performing classification. GSM [57] , KSVD [58] and BM3D [59] are well known algorithms for denoising and there have been several approaches using deep learning as a filter for denoising [60, 61]. Super resolution can also be applied as a pre-processing for LR images before image classification [62].

Another approach is to make image classifier robust against such image perturbation. [63] proposed fine-tuning a classifier with LR images after training the network with high resolution (HR) images. [64] proposed a two step training of partially coupled network for LR image classification. [65] studied the effect of noise and image quality degradation on the accuracy of a network, and proposed to re-train/fine-tune the network with data augmentation.

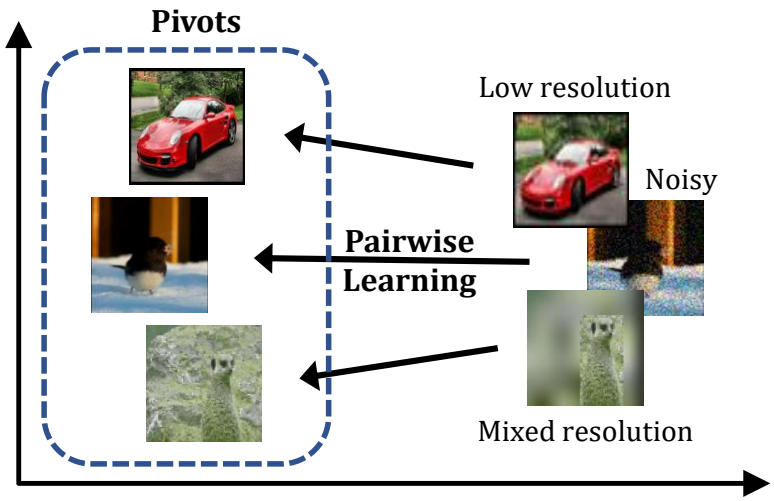


Figure 6.1: Use of embedding space for learning low-level similarities between clean and perturbed images.

This work falls in the latter category and makes the following contributions. First, utilization of embedding space for both image classification and *general pixel level regularization* is proposed. Inspired by the previous work [89] where adversarial noises are regularized with a unified embedding, the embedding space are used for regularizing general pixel level perturbations including LR, random noise and mixed resolution (MR) in a single image, which were not considered in the earlier study [89].

Second, unlike [63, 64] in which they focused on accuracy for LR images only, this study proposes to improve accuracy of a classifier *for perturbed images while maintaining decent accuracy for the original images*. The proposed method is applied for MNIST, CIFAR10 and ImageNet dataset considering various pixel perturbations and show improved accuracy compared to plain data augmentation approach.

6.2 Proposed Approach

Simple way to improve robustness against image perturbation is to train a classifier with perturbed images in addition to original images. This data augmentation approach is enhanced by adding simple, yet efficient regularization [89]. When training a classifier, k perturbed images are created using clean images and a mini-batch of size m is formulated

including both perturbed images and their corresponding clean images. The pairs of these images are injected during training. Un-normalized logits (embeddings) right before the softmax layer are used for both image classification and low-level pixel similarity learning. Standard cross entropy loss is used for image classification after the soft max layer and distance based loss is used for regularization of pixel value perturbation. In particular, the distance between clean and perturbed embeddings (resulted from the clean and perturbed images respectively) is minimized as shown in Figure 6.1. The intuition behind the use of distance based loss is to let the classifier aware of the pairs of two embeddings are from the same original images. This additional regularization promotes the classifier to ignore the pixel noises (difference between perturbed images and the original images). The entire procedure is described in algorithm 3.

The total loss if formulated as follows.

$$Loss = \frac{1}{(m-k)+\lambda k}(L_{org} + \lambda L_{perturb}) + \lambda_2 L_{dist} \quad (6.1)$$

where, L_{org} is the classification loss for the original images, $L_{perturb}$ is the classification loss for the perturbed images, L_{dist} is the distance based loss between the pairs of original and perturbed images. m is the size of the mini batch, k ($\leq m/2$) is the number of perturbed images in the mini batch. λ is the parameter to control the relative weight of the classification loss for perturbed images. λ_2 is the parameter to control the relative weight of the distance based loss L_{dist} in the total loss. Each loss term is defined as follows:

$$L_{org} = \sum_{i=1}^{m-k} L(\boldsymbol{X}_i|y_i)$$

$$L_{perturb} = \sum_{i=1}^{k} L(\boldsymbol{X}'_i|y_i)$$

$$L_{dist} = \sum_{i=1}^{k} ||\boldsymbol{E}'_i - \boldsymbol{E}_i||_2^2$$

Algorithm 3 Image classification with pixel level regularization
m: size of mini batch, k: size of perturbed images

Require: $k \leq m/2$
1: **repeat**
2: Get mini batch $B = \{X_1, ..., X_{m-k}\}$.
3: Generate k perturbed examples $\{X'_1, ..., X'_k\}$ from corresponding original examples $\{X_1, ..., X_k\}$.
4: Make new mini batch $B' = \{X'_1, ..., X'_k, X_1, ..., X_{m-k}\}$.
5: Perform one step forward pass with B'.
6: Formulate the cross entropy loss with entire embeddings $\{E'_1, ..., E'_k, E_1, ..., E_{m-k}\}$.
7: Formulate the distance based loss with perturbed embeddings $\{E'_1, ..., E'_k\}$ and corresponding original embeddings $\{E_1, ..., E_k\}$.
8: Perform one step backward pass and update the parameters in N.
9: **until** training converged

where, $\boldsymbol{X'_i}$ is i'th perturbed image generated from i'th original image $\boldsymbol{X}_i$. $\boldsymbol{E}_i$ and $\boldsymbol{E'_i}$ are the resulting embeddings from $\boldsymbol{X}_i$ and $\boldsymbol{X'_i}$ respectively. Pivot loss introduced in [89] is used where the gradient back propagation is not performed through original embeddings $\boldsymbol{E}$. Additional details can be found in [89].

6.3 Experimental Results

6.3.1 MNIST and CIFAR10

20-layer ResNet (Table 6 in [3]) model is used for MNIST and CIFAR10 dataset. The image values are scaled down to [0,1] for both dataset and subtracted by per-pixel mean values only for CIFAR10. 32x32 random crop and random flip are performed on zero padded 40x40 original images for CIFAR10. Networks are trained with λ = 0.3, λ_2 = 0.0001, m = 128, k = 64 for the experiments. Stochastic gradient descent (SGD) optimizer with momentum of 0.9, weight decay of 0.0001 are used. Training is started with a learning rate of 0.1. The learning rate is then divided by 10 at 4k and 6k iterations for MNIST, and 48k and 72k iterations for CIFAR10. Training is terminated at 8k iterations for MNIST, and 94k iterations for CIFAR10.

Table 6.1: Test accuracy (%) for Gaussian noise on MNIST and CIFAR10 dataset. (Clean, Noisy / Pivot loss) are trained with max_σ = 0.15. Higher accuracy among (Ours) and (Clean, Noisy) is emphasized in **bold**.

MNIST

Training	clean	σ = 0.15
Clean only	99.6	40.9
Noisy only	99.5	99.5
Clean, Noisy	**99.6**	99.4
Pivot loss (Ours)	99.5	**99.5**

CIFAR10

Training	clean	σ = 0.05	σ = 0.15
Clean only	91.6	62.4	16.6
Noisy only	89.8	88.9	82.0
Clean, Noisy	90.5	89.1	80.7
Pivot loss (Ours)	**90.8**	**89.6**	**81.7**

Random Noise

Gaussian noise is used as an example of random noise. During training, Gaussian noise $\mathcal{N}(\mu = 0, \sigma^2)$ is generated where σ is selected randomly in the interval [0, max_σ] per each image. The noises are added to the images, and the resulting images are clipped with the range [0,1]. Networks are trained by injecting those noisy images and clean images with/without pivot loss. Standard training with clean images and only with noisy images are included for comparison.

Table 6.1 shows test accuracy for MNIST and CIFAR10 dataset. As expected, injecting noisy images during training improves accuracy for noisy images at test time compared to training only with clean images (Clean only) for both MNIST and CIFAR10. For simple images like MNIST, the pivot loss doesn't show any meaningful difference compared to data augmentation approach (Clean, Noisy) or training with noisy images only case (Noisy only).

Training only with noisy images: for CIFAR10 dataset, training only with noisy im-

ages (Noisy only) shows best accuracy for noisy images with higher σ (σ=0.15), however, at the expense of decreased accuracy for the clean images. It is natural that the network trained with noisy images perform well for the noisy images, but not for the clean images. Careful selection of σ is necessary when Gaussian noise is used as a data augmentation for the clean images since larger noise actually decreases accuracy for the clean images.

Training with clean and noisy images: (Clean, Noisy / Pivot loss) show good compromise between training only with clean images (Clean only) and with noisy images (Noisy only). The pivot loss only increase accuracy for both clean and noisy images compared to pure data augmentation approach. This shows that additional loss with a unified embedding results in better regularization than training only with augmented data when dealing with pixel level perturbation.

Low Resolution

LR images are generated with sub-sampling factors of either 2 or 4. After down-sampling, those images are up-sampled with 'nearest' or 'bicubic' method. Randomly chosen sub-sampling factor and up-sampling method are used per each image during training. Resulting LR images together with the clean high resolution (HR) images are injected during training with/without pivot loss. Again, networks are trained with clean images and with LR images for comparison.

Table 6.2 shows test accuracy of the trained networks for clean and LR images. In this case as well, as expected, training with LR images improves accuracy for LR images compared to training only with clean images (Clean only) on both MNIST and CIFAR10.

Training only with LR images: for MNIST, training only with LR images (LR only) results in good accuracy for clean images as well as LR images. This is because MNIST LR images have good enough features that can be well generalized for clean HR images. For complex images like CIFAR10, however, training only with LR images (LR only) degrades accuracy for clean HR images since LR images lose features that are necessary

Table 6.2: Test accuracy (%) for LR images (sub-sampling = x4) on MNIST and CIFAR10 dataset. For CIFAR10 dataset, 110-layer ResNet models are also trained to analyze the effect of pivot loss for deeper networks. Higher accuracy among (Ours) and (Clean, LR) is emphasized in **bold**.

MNIST

Training	clean	x4 (nearest)	x4 (bicubic)
Clean only	99.6	50.4	57.5
LR only	99.5	87.6	86.4
Clean, LR	**99.6**	86.8	86.0
Pivot loss (Ours)	**99.6**	**87.1**	**86.2**

CIFAR10

Training	clean	x4 (nearest)	x4 (bicubic)
20-L, Clean only	91.6	19.2	19.9
20-L, LR only	86.9	75.9	72.9
20-L, Clean, LR	91.7	71.5	70.7
20-L, Pivot loss (Ours)	**92.0**	**73.5**	**72.7**
110-L, Clean only	93.5	21.9	15.7
110-L, LR only	86.4	78.8	77.0
110-L, Clean, LR	**93.8**	76.4	76.2
110-L, Pivot loss (Ours)	93.3	**78.1**	**77.4**

for HR image classification. In all cases, accuracy for LR images is not close to that for clean images due to the loss of features in LR images.

Training with clean and LR images: interestingly, networks trained with both clean and LR images (Clean, LR / Pivot loss) show better accuracy improvements for both clean and LR images compared to the network trained with clean images (Clean only). This suggests that data augmentation with LR images serves as a good regularizer for the clean images as well. However, (Clean, LR / Pivot loss) show decreased accuracy for LR images compared to (LR only). This is because the network trained only with LR images has seen more LR images during training, thus, performs better for LR images. Again, the pivot loss only increases accuracy for clean and LR images compared to the network trained without pivot loss.

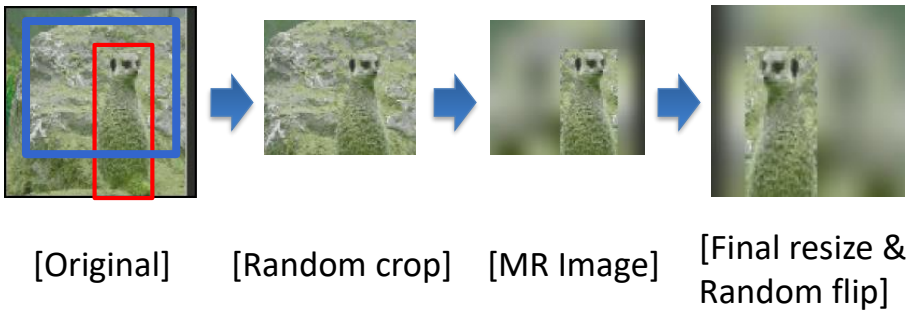


Figure 6.2: Generation of mixed resolution (MR) in a single image. Blue box and red box in the original image represent random crop area and bounding box respectively.

Deeper networks: Test the accuracies for 110-layer ResNet models on CIFAR10 dataset are presented to study the effect of training both with clean and LR images for deeper networks. As seen from the Table 6.2, data augmentation with 110-layer ResNet improves accuracy for both original and LR images compared to 20-layer counter part. The pivot loss increases accuracy for LR images compared to data augmentation approach, however, not for the clean images. The experiments show that the pivot loss is always a good regularizer for LR images, however, it is sometimes good or bad regularizer for the clean images. Thus, it is recommended to use distance based loss when the focus is to improve the accuracy of LR images while maintaining good enough accuracy for the original images.

6.3.2 ImageNet

18-layer ResNet (Table 1 in [3]) model is used for ImageNet [90] classification. The image values are scaled down to [0,1] and subtracted by per-channel mean values. Random crop ($0.5 <$ area range < 0.8) and random flip are performed on original images, and the resulting images are resized to 244x244. Training is started with a learning rate of 0.1. The learning rate is then divided by 10 at 500k, 800k and 1M iterations. Training is terminated at 1.1M iterations. First, a network is trained only with clean images.

Table 6.3: Test accuracy (%) for MR and LR images (sub-sampling = x4) on ImageNet dataset. Higher accuracy among (Ours) and (Clean, MR, LR) is emphasized in **bold**.

Training	clean	MR x4 (nearest)	LR x4 (nearest)
Clean only	68.1	57.8	29.3
Clean, MR, LR	**67.9**	61.5	44.9
Pivot loss (Ours)	67.4	**62.1**	**49.0**

Mixed Resolution in a Single Image

To see the effect of region of interest (RoI) based encoding on image classification, mixed resolution (MR) in a single image is considered as in Figure 6.2. Ground truth bounding boxes are used to create MR images for training. Region-based fully convolutional network (R-FCN) [91] with ResNet-101 trained on MS-COCO dataset [92] is used for bounding box generation on validation data set at test time. [1] Fining-tuning is performed on the baseline network with clean, MR and LR images for 400k iterations with a learnin rate of 1e-5 with/without pivot loss. λ = 0.3, λ_2 = 0.0001, m = 64 and k = 32 (16 for LR, 16 for MR) are used for the experiments.

Table 6.3 shows accuracy results for MR and LR images. Pivot loss shows better accuracy for MR and LR images compared to data augmentation approach (Clean, MR, LR) at the expense of small accuracy decrease on the clean images. This shows that pivot loss serves as a good regularizer for images with various resolution including MR and LR on ImageNet dataset. This result is also consistent with the LR case study for deeper networks on CIFAR10 dataset. As long as resolution and noise are concerned for image classification, the pivot loss can be a simple, yet efficient regularizer for perturbed images while maintaining decent accuracy for the clean images.

[1] Since ImageNet validation dataset doesn't have bounding boxes, R-FCN is used just for bounding box generation and observe decent quality of bounding box generation.

6.4 Summary

The use of embedding space for both image classification and pixel level regularization is proposed for various types of image perturbation. Image perturbation that can be happen in practical IoT scenario (random noise, LR and MR due to RoI based coding) are considered as examples of image perturbation. Image classification results on MNIST, CIFAR10 and ImageNet dataset showed promising results when proposed pivot loss is used as an additional regularization compared to the baseline data augmentation approach.

CHAPTER 7

NOISE-ROBUST AND RESOLUTION-INVARIANT OBJECT DETECTION

7.1 Introduction

Region of interest (RoI) based image processing enables efficient information retrieval and resource utilization on edge devices. The system takes inputs from sensors and processes the data to extract the meaningful information for pre-defined objectives. Recent years have shown exponential growth in spatial and temporal resolutions of image sensors, thanks to innovative technologies like digital pixels, resulting in large volume of sensor data [93]. The future advancements of sensors promise much higher image quality as well as unprecedented controllability of the camera parameters [56]. However, limited data transmission bandwidth between sensor and processor and available power creates a gap between maximum rates of data generation and consumption. Therefore, efficient control of sensors to manage the data generation while maintaining high quality of useful information is crucial for successful deployment of smart sensor nodes.

This chapter presents an adaptive camera module with deep neural network (DNN) based feedback control of sensor parameters to enhance quality of information content while limiting the sensor data. The adaptive camera module is used for real-time object detection and tracking system where the DNN-based feedback is used to locally adapt the spatial and/or temporal resolution of the image sensor to reduce maintain detection/tracking accuracy while using much lower volume of sensor data. However, when a vanilla object detection network is used for feedback, the multiple resolution and potentially, noisy images from the sensor leads to unstable behavior of the closed loop and degrade accuracy. To address this challenge, a light-weight mixture of pre-processing experts (MoPE) model is proposed as a pre-processing for object detection network. The variance of the input

images for the object detection network is minimized by applying different pre-processing experts for different images.

The prior efforts have proposed adaptive control of camera parameters such as exposure, gain, illumination, based on entropy, gradient etc. [94, 95, 96, 97]. However, the past efforts did not introduce DNN-based feedback in the control loop. Likewise, data augmentation and denoising techniques for robustness have been used, but at the expense of reduced detection accuracy for clean images [98]. Mixture of experts can be used with the gating network which discriminates input distributions and multiple object detection networks trained with images from different distributions [99, 100]. The model requires large memory for each expert, thus, it is impractical to use on the edge devices.

In contrast to the prior works, the research in this chapter makes following key contributions:

1. An adaptive camera algorithm is proposed for spatial/temporal resolution control based on DNN-based feedback enabling user-dependent determination of useful information.
2. The effect of pixel-level perturbation on the accuracy of DNN-based feedback control has been analyzed.
3. A robust object detection DNN using light-weight mixture of pre-processing experts has been proposed.
4. The proposed closed-loop system demonstrates high detection and tracking accuracy even under noise while managing data generation and transmission from the sensor.

Experimental results show that the proposed network achieves better detection/tracking accuracy for noisy images than the baseline network trained only with data augmentation, while maintaining accuracy under clean conditions.

7.2 Related Work

Camera parameter control: controlling camera parameters (like exposure or gain) based on image entropy was proposed in [94]. Shim et al. [95] used a gradient information based method to auto-adjust camera exposure. In [96], authors proposed an active control of illumination, shutter speed and voltage gain to improve the accuracy of object detection algorithms. In this chapter, a different approach that controls spatial and temporal resolution is applied to improve object detection and tracking accuracy while reducing the bandwidth usage. Wang et al. [97] recently proposed a two-stage framework with variable resolution control that uses low resolution image for object detection and high resolution image for object recognition.

Object detection with noisy images: several image denoising algorithms exist in the literature[101, 102, 103]. Milani et al. [104] proposed a much simpler adaptive filtering strategy for Histogram of Oriented Gradients(HOG) based object detection in noisy images. Recently, convolutional neural network (CNN) based object detection with noisy images is proposed in [105]. They used an adaptive bilateral filter that considers local texture properties to decide the amount of intensity smoothing required. They also showed the ability of CNN to adapt with image noise by re-training it with noisy images. However, they have not reported how this re-trained network performs with clean images with or without pre-processing. Applying pre-processing on every image has shown to degrade the detection accuracy for the clean images (Section 7.7). The proposed MoPE uses gating network which discriminates clean and noisy images and applies pre-processing only for noisy images.

7.3 Object Detection Background

Object detection is the problem of finding and classifying a number of objects in a single image. In contrast with problems like classification, the number of outputs can be varied

depending on how many objects are in an image. Types of the objects are not single one unlike classification problem. Multiple objects in multiple locations in an single images are need to be detected and classified.

Various methods use CNNs as backbone networks like image classification, however, many post processing elements are necessary to to produce set of bounding box and its corresponding class score. Faster R-CNN [106] is famous among the various object detection networks as it achieves state of the art detection performance and computationally efficient.

7.3.1 Faster R-CNN

Faster R-CNN uses unified feature extractor for both bounding box generation and image classification. Common choice of the feature extractor is CNN, and the output features are used in another CNN called region proposal network (RPN) to generate bounding box proposals. Based on the results from the RPN, the output of the feature extractor are cropped and fed into image classifier to get the class score.

7.3.2 Performance Metric

Performance measurement of the object detection can be done by calculating mean average precision (mAP) given fixed intersection over union (IoU). IoU can be calculated by:

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}},$$

where the area of overlap is computed between the predicted bounding box and the ground-truth bounding box, and area of union is the area encompassed by both the predicted bounding box and the ground-truth bounding box. The average precision for class c can be represented:

$$AP(c) = \frac{\#TP(c)}{\#TP(c) + \#FP(c)}$$

where the number of true positives, $\#TP(c)$, is the number of occurrences that the

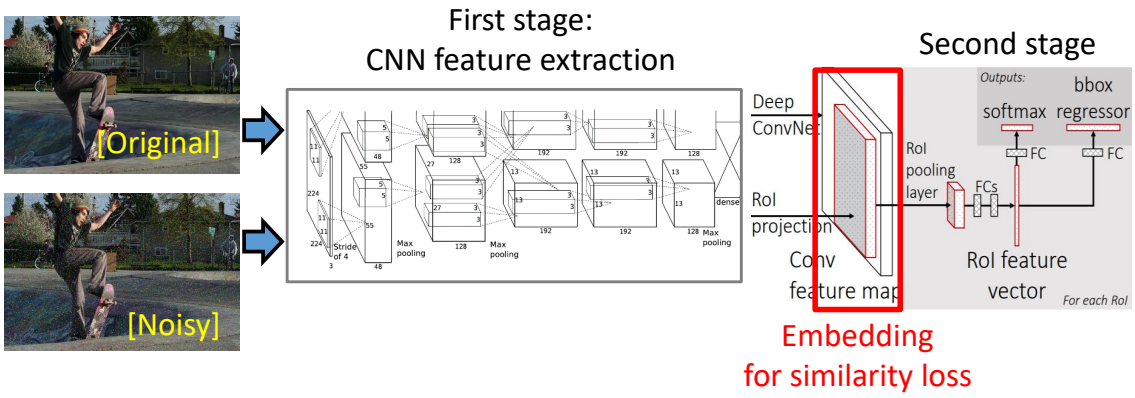


Figure 7.1: Faster R-CNN architecture [106] trained with similarity loss [89]

Table 7.1: Mean Average Precision @ IoU=0.5 on MS-COCO 2014 validation dataset. Faster R-CNN architecture [106] with inception v2 [107] as a backbone network is used. σ : standard deviation for Gaussian noise when the image values are in [0,1]. max_σ = 0.15 is used during training

		mAP @ IoU=0.5	
Training	Similarity loss	Clean	$\sigma = 0.15$
Clean+Noisy	-	0.458	0.305
Clean+Noisy	✓	0.457	0.306

proposal was made for class c and there actually was an object of class c. The number of false positives, $\#FP(c)$, is the number of occurrences that the proposal was made for class c and there is no object of class c. Objectness in the predicted bounding box is determined using IoU threshold as defined above. And the mAP is calculated by:

$$mAP = \frac{1}{|classes|} \sum_{c \in classes} AP(c)$$

7.4 Effect of low level similarity learning for Faster R-CNN

Similarity learning as in Chapter 6 can also be applied to object detection problem. Figure 7.1 shows how to apply similarity loss for Faster R-CNN network as an example object detection network. As shown in the figure, the similarity loss can be applied to the last

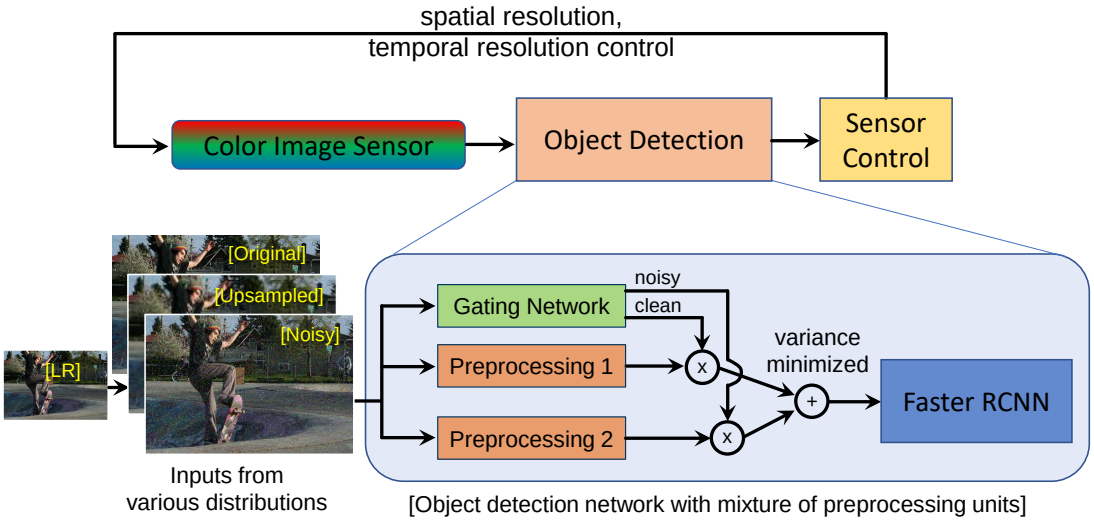


Figure 7.2: Object detection using active control of camera parameters

layer of the first feature extractor. The output of the first feature extractor is then used for both region proposal network and image classifier.

To compare the effect of similarity loss for the object detection network, two faster R-CNN networks have been trained with and without similarity loss. Table 7.1 shows that the difference between mAPs for those networks is negligible. The results differ from the results for the image classification task in Chapter 6. Possible explanation would be that the similarity loss is effective only for the continuous variables and not for the discrete variables. Even though region proposal network uses the same features with the image classifier, generation of bounding boxes relates with many discrete components including anchors and number of region proposals etc. And the object detection accuracy is determined based on the results from both region proposal network and image classifier.

7.5 Proposed Approach

The proposed object tracking system with image sensor, object detection network and feedback control unit is shown in Figure 7.2. The objective of the system is to extract the meaningful information for the target application (in this case, object tracking) from the

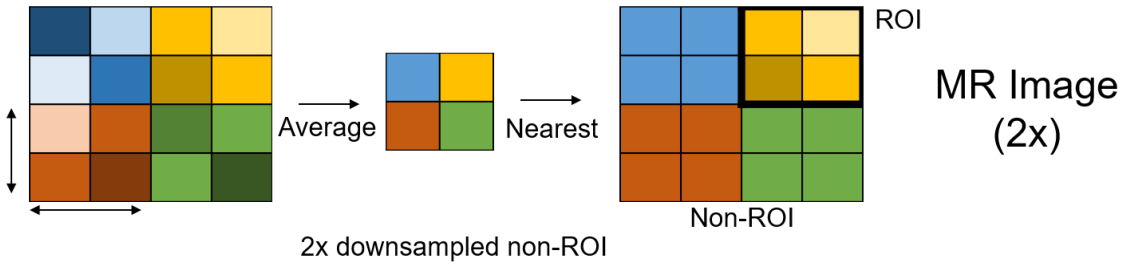


Figure 7.3: Mixed Resolution Sensor Model

controllable image sensor with optimized system bandwidth. Once an image from the sensor is obtained, object detection network produces the bounding boxes and confidence scores for each class. The control unit changes the spatial and temporal resolution of the image for the next frame. To be stable, enhancing the robustness of the object detection network for the noisy and low resolution images is necessary.

7.5.1 Spatio-Temporal Resolution Control

Sensor parameters in the camera is controlled to optimize the ratio of information to raw data sent by the sensor to the end application. An object detection network performs detection on the incoming image frames. Spatial resolution of the subsequent frames is then controlled by keeping regions of the image containing ROI at a higher resolution and decreasing resolution for non-ROI regions. Decreasing quality in the non-ROI regions is modeled as an averaging operation with the same pixel value copied for a specified grid of N x N pixels. The mixed resolution image model is presented in Figure 7.3.

Due to non-zero latency between feedback applied at the sensor and feedback generation from the network, there is a chance that objects under observation may fall partially under low resolution regions or be completely missed in the subsequent frames. An ROI prediction module is added to offset the detections from the object detection network to compensate for these latencies. Object tracking algorithm presented by Bewley et al is utilized for the ROI prediction problem [108].

The ROI prediction module uses multiple Kalman filters with a linear motion model.

Each of these predictors correspond to a unique detection from the object detection module. They are used for predicting likely locations of the object in the successive frames. Since the output of the object detector is stateless, an association module associates past predictors with current detections. For the association module, The Hungarian algorithm is used for bipartite graph matching with the IoU between predictions and detections as the cost for optimization. The ROI prediction module allows recovery in case of missed detections in between frames and ensures that objects of interest are always transmitted at the highest quality.

Using the same framework as above, temporal resolution control is done by using the output of the object detection itself to guide the acquisition of input images. Sequences are sampled at a reduced frame rate when no objects are detected or no prior objects are being tracked. Upon detection of objects of interest, spatial location containing object of interest are sampled at a higher rate. The proposed approach enables to achieve similar tracking accuracy while operating at a much reduced bandwidth conceivably conserving sensor power and bandwidth between the sensor and the rest of the system.

7.5.2 Robust Object Detection

Reasonable amount of image distortion is useful as a regularization when the test data distribution is expected to be in the same distribution with the training data. In this work, the input distribution is diverse including various resolutions and noisy images. Thus, data augmentation increases the overall detection accuracies, however, it reduces individual accuracy compared to the network trained only on the specific data distribution. As shown in Table 7.2, the network trained only with the clean images perform poorly on the low resolution and noisy images. The network trained with clean, low resolution and noisy images improves detection accuracy for low resolution and noisy images compared to the baseline at the expense of decreased accuracy on the clean images.

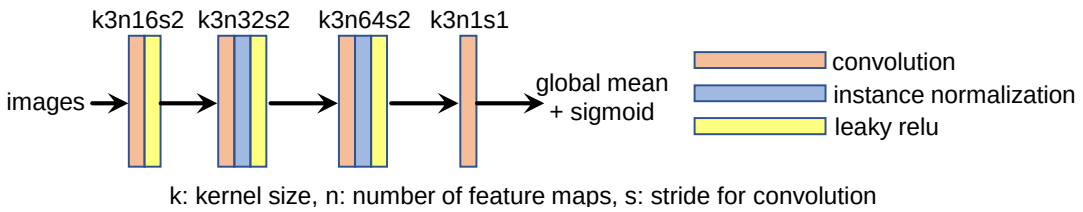


Figure 7.4: Gating network architecture

Mixture of pre-processing experts (MoPE)

To tackle this problem, a light-weight MoPE model is used to incorporate various input distributions. Each pre-processing expert is trained on the target input distribution like low resolution or noisy images and used as a pre-processing for the object detection network. Pre-processing unit trained on each data distribution minimizes the variance of the input distribution seen by the object detection network. A gating network is used to discriminate the input distribution whether the image is clean or noisy as shown in Figure 7.2. The gating network selects an input image for the object detection network among the output candidate images from pre-processing units.

Gating network: 31x31 modified PatchGAN [109] is used for the gating network as Gaussian noise in nature is easy to identify in a relatively small region. As shown in Figure 7.4, 3x3 kernels for convolution layers and instance normalization [110] are used as building blocks. Shallower network is used to minimize the burden of using pre-processing.

Pre-processing for the clean and low-resolution images: No preprocessing is used for the high-resolution images and up-sampled version of low resolution images. As the objective of using the mixture of the preprocessing units was to develop a computationally efficient architecture, data augmentation was sufficient to enhance the robustness against high-resolution and low-resolution images.

Pre-processing for the noisy images: An average filter and denoise auto-encoder network have been considered as a candidate pre-processing for the noisy images. Average filter was very powerful preprocessor for the noisy images considering its simpleness, thus, included in this experiments. U-Net like architecture is used for denoise neural net-

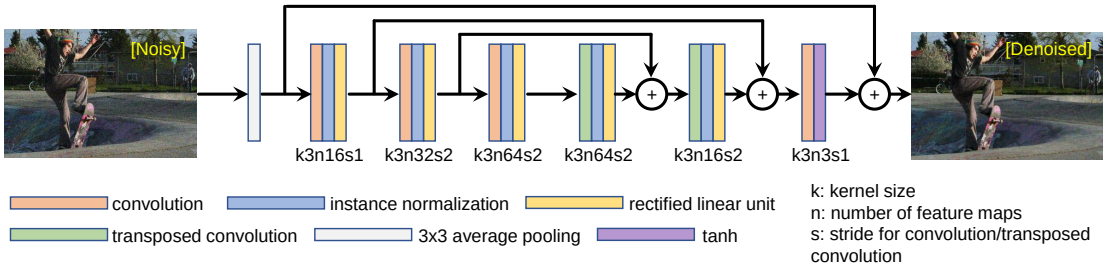


Figure 7.5: Denoise network architecture

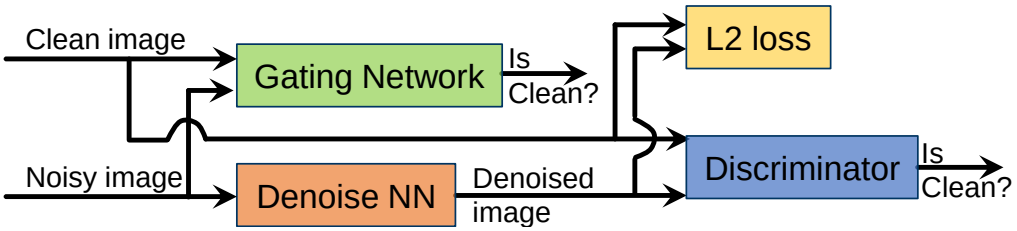


Figure 7.6: Training framework for denoise network with similarity and adversarial loss

work [111] as shown in Figure 7.5. Simple architecture is used intentionally to reduce the computational complexity. All the convolution and transposed convolution have 3x3 kernels, and the number of filters increases/decreases by 2 when the features are down/up sampled with stride 2. As the denoise network is pixel to pixel transformation between similar images, skip connection helps preserve the color information in original images, thus, eventually produce better quality. Average filter in front reduces the random noise in images at the expense of loss the edge information. [1] Loss of the edge information is recovered by training denoise network with adversarial loss [112] which will be discussed in the following section.

Training objectives

Denoise network: Adversarial loss [112] is used on top of the similarity loss (L2 loss) as shown in Figure 7.6 for realistic denoising. For the noise function $F : X \rightarrow Y$ and denoise

[1] Denoise network without average filter showed slight decreased accuracy.

function $G : Y \rightarrow X$ and its discriminator D, the objective is:

$$\mathcal{L}_{\text{GAN}}(G, D|F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log(1 - D(G(y)))], \tag{7.1}$$

where G aims to generate images $G(y)$ that look similar to images from clean data distribution X, while D tries to distinguish between denoised images $G(y)$ and the original images x for a given noise function $y = F(x)$.

Similarity loss is added to encourage the denoised images $G(F(x))$ to be similar to the original images x:

$$\mathcal{L}_{\text{sim}}(G|F) = ||G(F(x)) - x||_2^2. \tag{7.2}$$

The total objective is:

$$\mathcal{L}(G, D|F) = \mathcal{L}_{\text{GAN}}(G, D|F) + \lambda \mathcal{L}_{\text{sim}}(G|F), \tag{7.3}$$

where λ controls the relative importance of the two objectives. λ = 1 is used in the following experiments. Finally, the following optimization problem is solved:

$$G^* = \arg \min_G \max_D \mathcal{L}(G, D|F). \tag{7.4}$$

Gating network: Softmax function is used for the gating network as this is the classification problem. For the gating function H, the original images x and the noisy images $F(x)$, the training objective is:

$$\mathcal{L}_{\text{Gate}}(H) = -\log(H(x)) - \log(1 - H(F(x))), \tag{7.5}$$

where the output function of H is sigmoid function. The output of H is directly used to select the input images for the object detection network among the candidate images from

pre-processing experts.

7.6 Implementation

Faster R-CNN [106] is used for object detection network and inception v2 [107] as a backbone feature extractor. Tensorflow object detection API [113] is used and modified to integrate MoPE model. MS-COCO 2014 dataset [92] is used for training and follow the default configuration except the learning rate schedule in [113].

Up-sampled version of the 2x and 4x low-resolution images and noisy images along with the clean images are used as a data augmentation. Gaussian noise $\mathcal{N}(\mu = 0, \sigma^2)$ is used where σ is selected randomly in the interval [0, max_σ] per each image. For every iteration, only one image is selected randomly between the clean and the low-resolution image. The selected image is injected with the corresponding Gaussian noise added image.

Faster R-CNN is trained with stochastic gradient descent (SGD) optimizer with momentum of 0.9 for 600k iterations. Training is started with a learning rate of 2e-4 and divide it by 10 at 200k, 400k iterations. Denoise network and gating network is trained with Adam optimizer for 20k iterations. For fine-tuning of faster R-CNN and MoPE model, extra 200k iterations are used with a learning rate of 2e-6.

7.7 Experimental Results

To prove the effectiveness of the proposed approach, baseline network and network trained with data augmentation are used for comparison. As discussed, average filter and denoise network are considered for the pre-processing experts, and the effect of gating network is analyzed.

7.7.1 Object Detection

Object detection accuracies on MS-COCO validation dataset are shown in Table 7.2.

Table 7.2: Mean Average Precision @ IoU=0.5 on MS-COCO 2014 validation dataset. Faster R-CNN architecture [106] with inception v2 [107] as a backbone network is used. LR: low resolution, 2x and 4x down-sampled versions are used for training and 4x down-sampled images are used for evaluation. σ : standard deviation for gaussian noise when the image values are in [0,1]. max_σ = 0.15 is used during training

					mAP @ IoU=0.5		
Model	Training	Preprocessing	Gating	Fine-tune	Clean	LR	σ = 0.15
1	Clean only	-	-	-	**0.461**	0.335	0.226
2	Clean only	Average filter	-	-	0.445	0.336	0.299
3	Clean only	Average filter	✓	-	0.457	0.335	0.298
4	Clean only	Denoise	-	-	0.449	0.335	0.324
5	Clean only	Denoise	✓	-	0.457	0.335	0.323
6	Clean+LR+Noisy	-	-	-	0.454	0.360	0.295
7	Clean+LR+Noisy	Average filter	✓	✓	0.456	0.360	0.338
8	Clean+LR+Noisy	Denoise	✓	✓	0.456	0.360	**0.359**

Effect of pre-processing: The network trained only with the clean data shows reduced mAP for the noisy images (model 1). Pre-processing like average filter or denoise network (model 2 and model 4) increases mAP for the noisy images. Denoise network performs better than average filter at the cost of extra computation. For the clean images, however, pre-processing decreases mAP as additional processing on the clean images degrades input feature information.

Effect of gating network: Gating network trained in isolation is used together with pre-processing unit to analyze the effect of gating network. Model 3 and 5 in Table 7.2 show recovered mAP for the clean images as gating network guides the clean images not to be pre-processed. Eventually, the mAP for the clean images becomes similar with the baseline model (model 1).

Effect of MoPE: The proposed methods are compared with the network trained with data augmentation. The network trained with various input images show improved mAP for the low resolution and noisy images (model 6) compared to the baseline network (model 1).

(a) Clean image (b) Pre-processed clean image after MoPE (c) Noisy image (d) Pre-processed noisy image after MoPE

Figure 7.7: Sample images after the MoPE layer for the model 8 in Table 7.2

Next, the baseline network (model 1) together with pre-trained gating network and denoise network is fine-tuned. When fine-tuning, the loss functions used in training the gating network and the denoise network are not used. Classification and box regression losses are only used in fine-tuning. The same data augmentation used for training model 6 is applied.

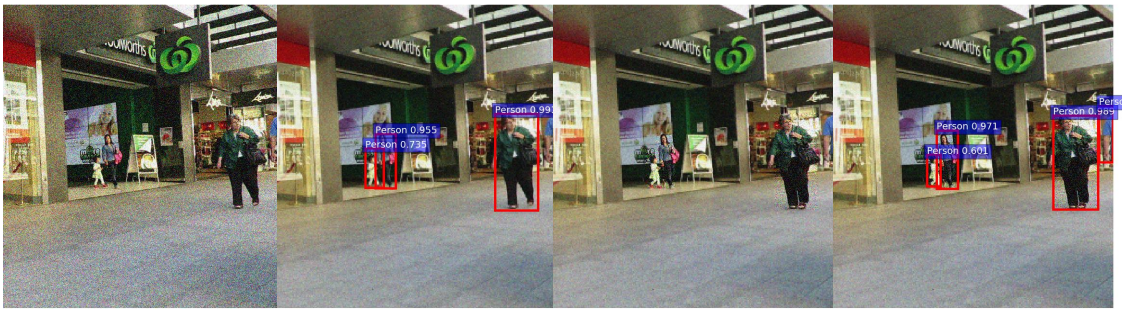
The proposed MoPE models (model 7 and 8) only improve mAP for the noisy images compare to the data augmented network (model 6). Gating network allows no-preprocessing for the clean and low-resolution images and injection of average filtered/denoised images during training further improves performance for the noisy images. Denoise network performs better than average filter with a small increase in computation. Sample images after the MoPE layer for the model 8 are shown in Figure 7.7. As intended, the clean image is not affected by the denoise network and the noisy image is denoised with the denoise network.

7.7.2 Object Tracking

Experimental evaluation was performed on the MOT-Challenge Dataset which is primarily targeted towards multi-object tracking [114]. Experiments on the train set which contains 7 sequences with varying lengths are performed. 6 sequences have resolution 1920 x 1080

Table 7.3: Tracking accuracy on MOT17 Train Set. Faster R-CNN [106] with inception v2 [107] backbone network

	MOTA		
Network	No Feedback	Feedback	Feedback + $\sigma = 0.15$
Model 1	**0.444**	0.438	0.373
Model 6	0.440	0.432	0.401
Model 7	0.442	0.437	0.405
Model 8	0.442	0.433	**0.428**



(a) Noisy image @ t=0 (b) Prediction with low resolution image @ t=0 (c) Mixed resolution image with feedback @ t=1 (d) Prediction with mixed resolution image @ t=1

Figure 7.8: Sample images with feedback control under noisy condition for the model 8 in Table 7.2

while one has a resolution of 640 x 480. It is a challenging dataset with multiple targets to track, occlusion effects, background clutter. 4 sequences are shot from a moving platform while the rest are filmed using static cameras.

All models in 3 different configurations are evaluated. In the baseline configuration, no feedback control is applied to modify the parameters of the video sequence. In the feedback configuration, spatio-temporal control is applied to sample the video at variable frame rates and variable resolution within the frame itself. In the feedback + noise configuration, Gaussian noise with $\sigma = 0.15$ is applied at every input sampled frame. For reproducibility and consistency across the models, same seed for the random number generation is used before each sequence. The main metric of interest is multiple object tracking accuracy (MOTA) and the average bandwidth consumed for transmitting the sequences.

Table 7.4: Bandwidth Consumption on MOT17 Train Set. Faster R-CNN [106] with inception v2 [107] backbone network

Average Bandwidth (Mbit/s)			
Network	No Feedback	Feedback	Feedback + σ = 0.15
Model 1	1311	378	333
Model 6	1311	382	363
Model 7	1311	379	377
Model 8	1311	374	383

Object tracking accuracy: The MOTA metric is a combined metric for evaluating three types of tracking errors i.e. false negatives, false positives and ID switches normalized by the number of ground truths within the sequence.

$$MOTA = (1 - \frac{\sum_i (fn_i + fp_i + id_i)}{\sum_i g_i}) * 100 \tag{7.6}$$

MOTA results are presented in Table 7.3. Feedback control slightly degrades MOTA results for all the cases. When Gaussian noise is added during the feedback control, MOTA results further drops. Network trained with data augmentation (model 6) shows better MOTA than the baseline. And the proposed method (model 7 and 8), especially for the model 8 with denoise network, shows better MOTA compared to other models for feedback control and noisy condition. The results are consistent with the results in object detection, and this proves the proposed approaches are beneficial for both object detection and tracking. Sample images are shown for the model 8 in Figure 7.8.

Object tracking bandwidth analysis: For the bandwidth calculation, the raw pixel count per frame is used and image/video compression techniques are used. Bandwidth results are presented in Table 7.4. In case of feedback control without noise, the average bandwidth can be reduced by 3x compared to the no-feedback case. Note, that even though all models show similar reduced bandwidth results for feedback case, MOTA results show that comparable tracking accuracy is achieved compared to no-feedback case by data aug-

mentation and the proposed methods.

For the feedback control in noisy condition, model 1 shows further decreased bandwidth than feedback case due to increased false negatives. The increased false negatives degrade the tracking accuracy as shown in Table 7.3. The proposed method (model 8) maintains similar bandwidth with that of feedback without noise condition while achieving comparable tracking accuracy.

7.8 Summary

In this chapter, efficient object detection and tracking system using the feedback control of the image sensor is presented. Noise-robust object detection network is realized with MoPE model for reliable feedback control under noisy condition. Feedback control enables reduced bandwidth allowing energy efficient operation of the system. Low resolution and noise robust object detection network is proposed using simple yet efficient mixture of pre-processing experts (MoPE) model. The overhead of adding gating network and denoise network is negligible considering heavy computation of object detection network. The proposed method can achieve better detection and tracking accuracy than the baseline network trained with data augmentation or the network attached with pre-processing modules.

CHAPTER 8
CONCLUSION

This chapter summarizes the key contributions and provides future research direction.

In this thesis, techniques and algorithms have been presented to enable energy efficient, secure and noise-robust deep learning for the internet of things (IoTs). Dynamic precision scaling (DPS) and flexible multiply-accumulator (MAC) units for speeding up convolutional neural networks (CNNs) training have been proposed. Low precision training for recurrent neural networks (RNNs) has been enabled by fine-tuning of hyper parameters. Cascade adversarial training and low-level similarity learning have been proposed to enhance robustness against adversarial attacks. Noise-robust and resolution-invariant image classification has been realized by applying pixel level similarity learning, and mixture of pre-processing experts (MoPE) model has been proposed to enable noise-robust object detection.

Chapter 3 presented an algorithm to speed up CNNs training by applying DPS. DPS utilizes dynamic fixed point format as a compromise between traditional fixed point and floating point format. DPS tries to find good enough low precision for training, not necessarily minimum precision for given network and data set. Once the integer part bit width allocation is done for each layer and neuron, DPS start finding good enough low precision by aggressively lowering precision. Moving average loss is used to determine dynamic precision search. Precision flexible MAC has been presented to show feasibility of speeding up of CNNs training when it is used together with DPS. Future work will include entire system level analysis for the real world platform to estimate actual benefit of using DPS and and flexible MAC.

Chapter 4 introduced comparative study of entire limited precision training of RNNs. Performance analysis of low precision hardware for RNN has also been presented. The key

observations from this study include: (1) batch normalization is essential even for RNNs training, (2) 64-bit fixed point is not sufficient for training RNNs, (3) careful selection for the overflow rate is critical when using dynamic fixed point format, (4) stochastic rounding is powerful for low precision training, (5) gradient accumulation for the weights if more important than any other path, (6) piecewise linear activation together with stochastic rounding works pretty well. The low precision MAC with stochastic rounding has been implemented and showed 4.7x faster and 4.55x energy efficient training. The future work would be to extend this work to generalize for deeper layer RNNs training.

In chapter 5, techniques to enhance robustness against adversarial attacks have been presented. Unified embedding space is utilized for both classification and low-level (pixel-level) similarity learning to ignore unknown pixel level perturbation. During training, adversarial images are injected without replacing their corresponding clean images and the distance between the two embeddings (clean and adversarial) is penalized. This additional regularization encourages two similar images (clean and perturbed versions) to produce the same outputs, not necessarily the true labels, enhancing classifier's robustness against pixel level perturbation. The study also showed that iteratively generated adversarial images easily transfered between networks trained with the same strategy. Inspired by this observation, cascade adversarial training has been proposed which transfers the knowledge of the end results of adversarial training. Cascade adversarial training injects iteratively generated adversarial images crafted from already defended networks in addition to one-step adversarial images from the network being trained. Experimental results showed that cascade adversarial training together with the proposed low-level similarity learning efficiently enhanced the robustness against iterative attacks, but at the expense of decreased robustness against one-step attacks. The study showed that combining those two techniques could also improve robustness under the worst case black box attack scenario. However, there is still a gap between accuracy for the clean images and that for the adversarial images. Improving robustness against both one-step and iterative attacks still remains challenging since

it is shown to be difficult to train networks robust for both one-step and iterative attacks simultaneously. Future research is necessary to further improve the robustness against iterative attack without sacrificing the accuracy for step attacks or clean images under both white-box attack and black-box attack scenarios.

In chapter 6, an algorithm for noise-robust and resolution-invariant image classification has been proposed. Pixel-level pair-wise similarity learning has been applied to enhance the classification accuracy for both the clean and perturbed images. The proposed approach uses pair of clean and perturbed images during training, and minimize the distance between the two embeddings. Gaussian noise added images, low resolution or mixed resolution images are studied as examples of pixel level perturbation to prove the effectiveness of the algorithm. Image classification results on MNIST, CIFAR10 and ImageNet dataset showed promising results when the networks are trained with the proposed approach. Still there is a gap between the accuracy for the clean images and that for the perturbed images, thus, the future work is necessary to further enhance the accuracy for the perturbed images without sacrificing accuracies for the perturbed images.

Chapter 7 presented techniques to enable noise-robust and resolution-invariant object detection. To this end, light weight MoPE model has been proposed to decrease the input variance seen by the object detection network. The data distribution for the original images and that for the noisy images become similar by applying specific pre-processing expert per each distribution. In particular, for noisy images, adversarially regularized auto-encoder network is used to denoise Gaussian noise added in the images. Skip connections in the network helps preserve the color information in the original images. Adversarial loss and the L2 distance loss are used to train the network. Fine-tuning the MoPE model and object detection network further improved the detection and tracking accuracy. Future work is necessary to further enhance detection accuracy for both clean and perturbed images.

Future of deep learning and the IoT is very promising, however, there are many challenges yet to be addressed. Today's deep learning advances have promised that getting

more data is crucial for the success of deep learning. However, collecting the data is not that simple. The data is distributed in an unorganized way and the privacy concerns prohibit the data collection from the edge devices. Use of edge devices for training might be the possible solution to address this challenge. Efficient training on the edge devices, thus, has to be studied to enable better model while preserving the user privacy.

Also, deep learning no longer has to be treated as a black box. Deep learning deployment has to be careful especially for the safe critical applications like autonomous driving car. Adoption of deep learning might be hindered if the technology doesn't fully explain the underlying mechanism. Understandable, robust and uncertainty reduced deep learning has to be developed to enable artificial intelligence for the safe critical application.

Appendices

APPENDIX A

IMPLEMENTATION DETAILS IN CHAPTER 5

A.1 Model Descriptions

The model names used in Chapter 5 are summarized in Table A.1. For ensemble adversarial training, pre-trained networks as in Table A.3 together with the network being trained are used to generate one-step adversarial examples during training. For cascade adversarial training, pre-trained defended networks as in Table A.2 are used to generate iter_FGSM images, and the network being trained is used to generate one-step adversarial examples during training.

Table A.1: Model descriptions

Dataset	ResNet	Initialization Group	Training	Model
MNIST	20-layer	A	standard training	R20M
			$\boldsymbol{K}$urakin's	R20M$_{\boldsymbol{K}}$
			$\boldsymbol{B}$idirection loss	R20M$_{\boldsymbol{B}}$
			$\boldsymbol{P}$ivot loss	R20M$_{\boldsymbol{P}}$
CIFAR10	20-layer	B	standard training	R20
			$\boldsymbol{K}$urakin's	R20$_{\boldsymbol{K}}$
			$\boldsymbol{E}$nsemble training	R20$_{\boldsymbol{E}}$
			$\boldsymbol{B}$idirection loss	R20$_{\boldsymbol{B}}$
			$\boldsymbol{P}$ivot loss	R20$_{\boldsymbol{P}}$
			$\boldsymbol{K}$urakin's & $\boldsymbol{C}$ascade training	R20$_{\boldsymbol{K,C}}$
			$\boldsymbol{P}$ivot loss & $\boldsymbol{E}$nsemble training	R20$_{\boldsymbol{P,E}}$
			$\boldsymbol{P}$ivot loss & $\boldsymbol{C}$ascade training	R20$_{\boldsymbol{P,C}}$
		C	standard training	R20$_{\mathbf{2}}$
			$\boldsymbol{K}$urakin's	R20$_{\boldsymbol{K}\mathbf{2}}$
			$\boldsymbol{P}$ivot loss	R20$_{\boldsymbol{P}\mathbf{2}}$
		D	standard training	R20$_{\mathbf{3}}$
		E	standard training	R20$_{\mathbf{4}}$
	56-layer	F	$\boldsymbol{K}$urakin's	R56$_{\boldsymbol{K}}$
			$\boldsymbol{P}$ivot loss	R56$_{\boldsymbol{P}}$
		G	$\boldsymbol{K}$urakin's	R56$_{\boldsymbol{K}\mathbf{2}}$
	110-layer	H	standard training	R110
			$\boldsymbol{K}$urakin's	R110$_{\boldsymbol{K}}$
			$\boldsymbol{P}$ivot loss	R110$_{\boldsymbol{P}}$
			$\boldsymbol{E}$nsemble training	R110$_{\boldsymbol{E}}$
			$\boldsymbol{K}$urakin's & $\boldsymbol{C}$ascade training	R110$_{\boldsymbol{K,C}}$
			$\boldsymbol{P}$ivot loss & $\boldsymbol{E}$nsemble training	R110$_{\boldsymbol{P,E}}$
			$\boldsymbol{P}$ivot loss & $\boldsymbol{C}$ascade training	R110$_{\boldsymbol{P,C}}$
		I	standard training	R110$_{\mathbf{2}}$
			$\boldsymbol{K}$urakin's	R110$_{\boldsymbol{K}\mathbf{2}}$
			$\boldsymbol{E}$nsemble training	R110$_{\boldsymbol{E}\mathbf{2}}$
			$\boldsymbol{P}$ivot loss	R110$_{\boldsymbol{P}\mathbf{2}}$
			$\boldsymbol{K}$urakin's & $\boldsymbol{C}$ascade training	R110$_{\boldsymbol{K,C}\mathbf{2}}$
			$\boldsymbol{P}$ivot loss & $\boldsymbol{E}$nsemble training	R110$_{\boldsymbol{P,E}\mathbf{2}}$
			$\boldsymbol{P}$ivot loss & $\boldsymbol{C}$ascade training	R110$_{\boldsymbol{P,C}\mathbf{2}}$
		J	standard training	R110$_{\mathbf{3}}$
		K	standard training	R110$_{\mathbf{4}}$

Table A.2: Ensemble model description

Ensemble models	Pre-trained models
R20$_{\boldsymbol{E}}$, R20$_{\boldsymbol{P,E}}$, R110$_{\boldsymbol{E}}$, R110$_{\boldsymbol{P,E}}$	R20$_{\mathbf{3}}$, R110$_{\mathbf{3}}$
R110$_{\boldsymbol{E2}}$, R110$_{\boldsymbol{P,E2}}$	R20$_{\mathbf{4}}$, R110$_{\mathbf{4}}$

Table A.3: Cascade model description

Cascade models	Pre-trained model
R20$_{\boldsymbol{K,C}}$, R20$_{\boldsymbol{P,C}}$	R20$_{\boldsymbol{P}}$
R110$_{\boldsymbol{K,C}}$, R110$_{\boldsymbol{P,C}}$	R110$_{\boldsymbol{P}}$
R110$_{\boldsymbol{K,C2}}$, R110$_{\boldsymbol{P,C2}}$	R110$_{\boldsymbol{P2}}$

A.2 Additional Black Box Attack Results

Table A.4: CIFAR10 test results (%) under black box attacks between the network with the same initialization (ϵ=16)}

Target	Source: step_FGSM			Source: iter_FGSM		
	R20	R20$_K$	R20$_P$	R20	R20$_K$	R20$_P$
R20	**12.2**	27.4	27.5	0.0	45.9	44.7
R20$_K$	**65.7**	81.5	81.8	51.5	0.0	**18.2**
R20$_P$	**58.1**	89.3	91.7	48.9	**13.4**	0.0

Table A.4 shows that black box attack between trained networks with the same initialization tends to be more successful than that between networks with different initialization as explained in [24].

Table A.5: CIFAR10 test results (%) under black box attacks for ϵ=16. {Target and Source networks are switched from the Table 5.1}

Target	Source: step_FGSM			Source: iter_FGSM		
	R20	R20$_K$	R20$_P$	R20	R20$_K$	R20$_P$
R20$_2$	**17.9**	33.9	34.5	**4.1**	54.8	54.3
R20$_{K2}$	**65.0**	84.6	84.5	61.2	**25.3**	**30.4**
R20$_{P2}$	**66.4**	88.2	87.2	61.6	**27.7**	**36.1**

In Table A.5, the proposed method (R20$_{P2}$) is always better at one-step and iterative black box attack from defended networks (R20$_K$, R20$_P$) and undefended network (R20) than Kurakin's method (R20$_{B2}$). However, it is hard to tell which method is better than the other one as explained in the main paper.

In Table A.6, black box attack accuracies are shown with the source and the target networks switched from the Table 5.6. Similar observation is found that networks trained with both low-level similarity learning and cascade/ensemble adversarial training (R110$_{P,C2}$, R110$_{P,E2}$) show better *worst-case* performance than other networks. Overall, iter_FGSM

Table A.6: CIFAR10 test results (%) for cascade networks under black box attacks for ϵ=16. {Target and Source: Please see the model descriptions in Appendix A.1.}

Target	Source: iter_FGSM						
	R110	R110$_{\boldsymbol{K}}$	R110$_{\boldsymbol{E}}$	R110$_{\boldsymbol{P}}$	R110$_{\boldsymbol{K,C}}$	R110$_{\boldsymbol{P,E}}$	R110$_{\boldsymbol{P,C}}$
R110$_{\boldsymbol{K2}}$	80.5	72.7	49.3	68.0	49.6	**41.0**	67.9
R110$_{\boldsymbol{E2}}$	82.7	59.1	**39.5**	59.6	51.5	40.3	69.6
R110$_{\boldsymbol{P2}}$ (Ours)	80.3	75.9	54.2	72.2	54.9	**44.3**	72.4
R110$_{\boldsymbol{K,C2}}$ (Ours)	62.1	74.7	61.5	72.3	46.5	**39.0**	67.9
R110$_{\boldsymbol{P,E2}}$ (Ours)	81.5	79.0	50.0	77.3	56.9	**45.5**	75.2
R110$_{\boldsymbol{P,C2}}$ (Ours)	72.2	76.4	60.6	73.8	51.0	**40.9**	72.0

images crafted from ensemble model families (R110$_{\boldsymbol{E}}$, R110$_{\boldsymbol{P,E}}$) remain strong on the defended networks.

A.3 Implementation Details for Carlini-Wagner L_∞ Attack

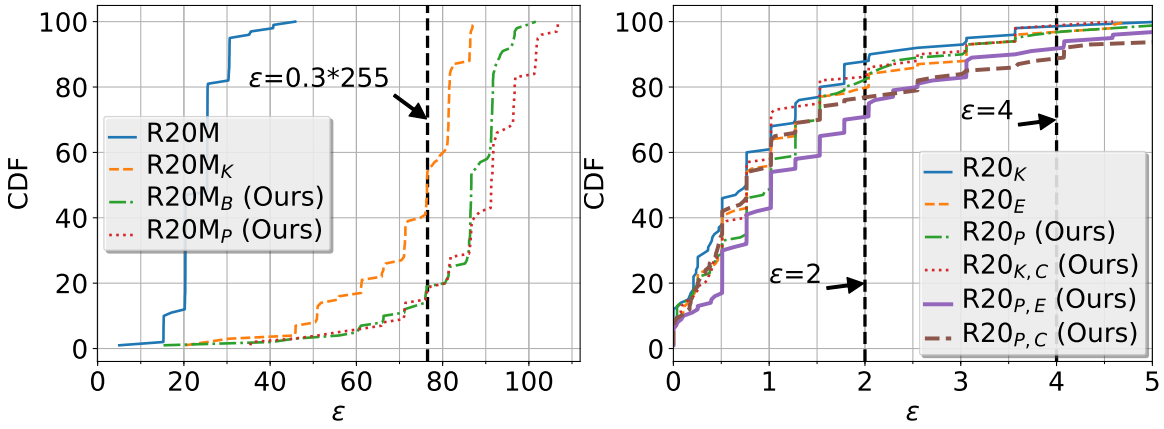
Carlini and Wagner (CW) L_∞ attack solves the following optimization problem for every input $\boldsymbol{X}$.

$$minimize \quad c \cdot f(\boldsymbol{X}+\boldsymbol{\delta}) + \sum_i [(|\delta_i| - \tau)^+]$$

$$such\ that \quad \boldsymbol{X}+\boldsymbol{\delta} \in [0,1]^n$$

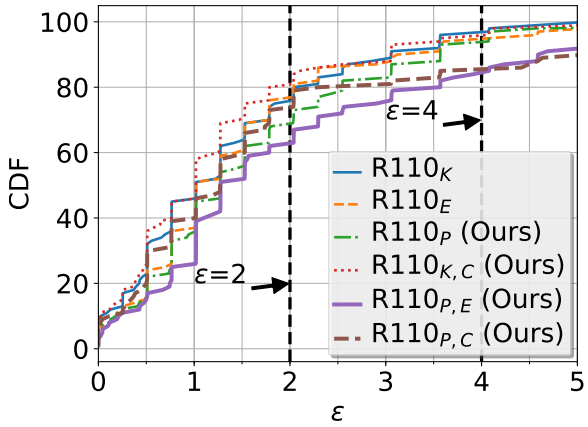
where, the function f is defined such that attack is success if and only if $f(\boldsymbol{X}+\boldsymbol{\delta}) < 0$, $\boldsymbol{\delta}$ is the target perturbation defined as $\boldsymbol{X}^{adv} - \boldsymbol{X}$, c is the parameter to control the relative weight of function f in the total cost function, and τ is the control threshold used to penalize any terms that exceed τ.

Since CW L_∞ attack is computationally expensive, 100 test examples are used (10 examples per each class). Adversarial examples $\boldsymbol{X}^{adv}$ are searched with $c \in \{0.1, 0.2, 0.5, 1, 2, 5, 10, 20\}$ and $\tau \in \{0.02, 0.04, ..., 0.6\}$ for MNIST and $c \in \{0.1, 0.3, 1, 3, 10, 30, 100\}$ and $\tau \in \{0.001, 0.002, ..., 0.01, 0.012, ..., 0.02, 0.024, ..., 0.04, 0.048, ..., 0.08\}$ for CIFAR10. Adam optimizer is used with an initial learning rate of $0.01/c$. Scaled version of initial learning



(a) MNIST

(b) CIFAR10, 20-layer



(c) CIFAR10, 110-layer

Figure A.1: Cumulative distribution function vs. ϵ for 100 test adversarial examples generated by CW L_∞ attack. Lower CDF value for a fixed ϵ means the better defense.

rate is used since the constant initial learning rate for $c \cdot f(\boldsymbol{X} + \boldsymbol{\delta})$ term is critical for successful adversarial images generation. The search is terminated after 2,000 iterations for each $\boldsymbol{X}$, c and τ. If $f(\boldsymbol{X} + \boldsymbol{\delta}) < 0$ and the resulting $||\boldsymbol{\delta}||_\infty$ is lower than the current best distance, $\boldsymbol{X}^{adv}$ is updated.

Figure A.1 shows cumulative distribution function of ϵ for 100 successful adversarial examples per each network. The number of adversarial examples with $\epsilon >$ 0.3*255 for MNIST and that with $\epsilon > 2$ or 4 for CIFAR10 are reported. As seen from this figure, the proposed approaches provide robust defense against CW L_∞ attack compared to other approaches.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 770–778.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[5] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, IEEE Computer Society, 2016, pp. 2818–2826.

[6] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[8] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, May 2011.

[9] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, 2015, pp. 2048–2057.

[10] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, and J. Makhoul, "Fast and robust neural network joint models for statistical machine translation," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguis-*

tics (Volume 1: Long Papers), Baltimore, Maryland: Association for Computational Linguistics, 2014, pp. 1370–1380.

[11] J. Chung, K. Cho, and Y. Bengio, "A character-level decoder without explicit segmentation for neural machine translation," in *ACL (1)*, The Association for Computer Linguistics, 2016.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 3104–3112.

[13] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. Hannun, T. Han, L. Johannes, B. Jiang, C. Ju, B. Jun, P. LeGresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, Y. Xie, D. Yogatama, B. Yuan, J. Zhan, and Z. Zhu, "Deep speech 2 : End-to-end speech recognition in english and mandarin," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., ser. Proceedings of Machine Learning Research, vol. 48, New York, New York, USA: PMLR, 2016, pp. 173–182.

[14] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *ICASSP*, IEEE, 2016, pp. 4960–4964.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[17] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network,"

in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed., Morgan-Kaufmann, 1990, pp. 396–404.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[19] K. Cho, B van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8), 2014*, 2014.

[20] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13, Atlanta, GA, USA: JMLR.org, 2013, pp. III–1310–III–1318.

[21] Y. Long, E. M. Jung, J. Kung, and S. Mukhopadhyay, "Reram crossbar based recurrent neural network for human activity detection," in *IJCNN*, IEEE, 2016, pp. 939–946.

[22] J. Ott, Z. Lin, Y. Zhang, S. Liu, and Y. Bengio, "Recurrent neural networks with limited numerical precision," *CoRR*, vol. abs/1608.06902, 2016.

[23] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[24] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[25] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvári, "Learning with a strong adversary," *CoRR*, vol. abs/1511.03034, 2015.

[26] J. H. Ko, T. Na, and S. Mukhopadhyay, "An energy-efficient wireless video sensor node with a region-of-interest based multi-parameter rate controller for moving object surveillance," in *Advanced Video and Signal Based Surveillance, (AVSS)*, 2016, pp. 138–144.

[27] J. H. Ko, M. F. Amir, K. Z. Ahmed, T. Na, and S. Mukhopadhyay, "A single-chip image sensor node with energy harvesting from a CMOS pixel array," *IEEE Trans. on Circuits and Systems I*, vol. 64-I, no. 9, pp. 2295–2307, 2017.

[28] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[29] J. H. Ko, D. Kim, T. Na, J. Kung, and S. Mukhopadhyay, "Adaptive weight compression for memory-efficient neural networks," in *Design, Automation & Test in Europe Conference & Exhibition, (DATE)*, 2017, pp. 199–204.

[30] M. Mathieu, M. Henaff, and Y. Lecun, "Fast training of convolutional networks through ffts," in *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[31] J. H. Ko, B. A. Mudassar, T. Na, and S. Mukhopadhyay, "Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation," in *Design Automation Conference, (DAC)*, 2017, 59:1–59:6.

[32] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, 2016.

[33] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.

[34] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *NIPS*, 2015, pp. 3123–3131.

[35] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15, Monterey, California, USA: ACM, 2015, pp. 161–170, ISBN: 978-1-4503-3315-3.

[36] A. Rahman, J. Lee, and K. Choi, "Efficient fpga acceleration of convolutional neural networks using logical-3d compute array," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, ser. DATE '16, Dresden, Germany: EDA Consortium, 2016, pp. 1393–1398, ISBN: 978-3-9815370-6-2.

[37] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16, Seoul, Republic of Korea: IEEE Press, 2016, pp. 243–254, ISBN: 978-1-4673-8947-1.

[38] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47, Cambridge, United Kingdom: IEEE Computer Society, 2014, pp. 609–622, ISBN: 978-1-4799-6998-2.

[39] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural

Networks," in *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, 2016, 262–263.

[40] D. Shin, J. Lee, J. Lee, and H. Yoo, "14.2 DNPU: an 8.1tops/w reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *ISSCC*, IEEE, 2017, pp. 240–241.

[41] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "Fpga-based accelerator for long short-term memory recurrent neural networks," in *ASP-DAC*, IEEE, 2017, pp. 629–634.

[42] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. B. J. Dally, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17, Monterey, California, USA: ACM, 2017, pp. 75–84, ISBN: 978-1-4503-4354-1.

[43] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 448–456.

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.

[45] S. Han, J. Pool, S. Narang, H. Mao, S. Tang, E. Elsen, B. Catanzaro, J. Tran, and W. J. Dally, "DSD: regularizing deep neural networks with dense-sparse-dense training flow," *CoRR*, vol. abs/1607.04381, 2016.

[46] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 1737–1746.

[47] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv e-prints*, vol. abs/1412.7024, Dec. 2014.

[48] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1223–1231.

[49] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proceedings of*

the 11th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'14, Broomfield, CO: USENIX Association, 2014, pp. 571–582, ISBN: 978-1-931971-16-4.

[50] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time.," in *ECML/PKDD (3)*, ser. Lecture Notes in Computer Science, vol. 8190, Springer, 2013, pp. 387–402, ISBN: 978-3-642-40993-6.

[51] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[52] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," in *Proceeedings of the ACM Asia Conference on Computer and Communications Security (ASIACCS'17)*, 2017.

[53] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, "Transferability in machine learning: From phenomena to black-box attacks using adversarial samples," *CoRR*, vol. abs/1605.07277, 2016.

[54] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Proceedings of the International Conference on Learning Representations (ICLR) Workshop*, 2017.

[55] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*, vol. 00, 2016, pp. 582–597.

[56] DARPA. (2016). A camera that can see unlike any imager before it, (visited on 09/16/2016).

[57] J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli, "Image denoising using scale mixtures of gaussians in the wavelet domain," *IEEE Trans. Image Processing*, vol. 12, no. 11, pp. 1338–1351, 2003.

[58] M. Elad and M. Aharon, "Image denoising via learned dictionaries and sparse representation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006, pp. 895–900.

[59] K. Dabov, A. Foi, V. Katkovnik, and K. O. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Trans. Image Processing*, vol. 16, no. 8, pp. 2080–2095, 2007.

[60] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 350–358.

[61] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising: Can plain neural networks compete with bm3d?" In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 2392–2399.

[62] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[63] X. Peng, J. Hoffman, S. X. Yu, and K. Saenko, "Fine-to-coarse knowledge transfer for low-res image classification," in *2016 IEEE International Conference on Image Processing, (ICIP)*, 2016, pp. 3683–3687.

[64] Z. Wang, S. Chang, Y. Yang, D. Liu, and T. S. Huang, "Studying very low resolution recognition using deep networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[65] Y. Zhou, S. Song, and N. Cheung, "On classification of distorted images with deep convolutional neural networks," in *International Conference on Acoustics, Speech and Signal Processing, (ICASSP)*, 2017, pp. 1213–1217.

[66] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *ICANN*, ser. Lecture Notes in Computer Science, vol. 8681, Springer, 2014, pp. 281–290.

[67] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," *CoRR*, vol. abs/1510.03009, 2015. arXiv: 1510.03009.

[68] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, 1998, pp. 2278–2324.

[69] D. Williamson, "Dynamically scaled fixed point arithmetic," in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 1991.

[70] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, New York, NY, USA: JMLR.org, 2016, pp. 2849–2858.

[71] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, ser. MM '14, Orlando, Florida, USA: ACM, 2014, pp. 675–678, ISBN: 978-1-4503-3063-3.

[72] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "Cudnn: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014.

[73] W. Chen, J. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing convolutional neural networks in the frequency domain," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: ACM, 2016, pp. 1475–1484, ISBN: 978-1-4503-4232-2.

[74] I. Laptev and T. Lindeberg, "Velocity adaptation of space-time interest points," in *ICPR (1)*, IEEE Computer Society, 2004, pp. 52–56.

[75] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. J. Goodfellow, A. Bergeron, and Y. Bengio, "Theano: Deep learning on gpus with python," in *Big Learn workshop, NIPS'11*, 2011.

[76] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch normalized recurrent neural networks," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 2657–2661.

[77] T. Cooijmans, N. Ballas, C. Laurent, C. Gulcehre, and A. C. Courville, "Recurrent batch normalization," 2017.

[78] T. Na and S. Mukhopadhyay, "Speeding up convolutional neural network training with dynamic precision scaling and flexible multiplier-accumulator," in *International Symposium on Low Power Electronics and Design, (ISLPED)*, 2016, pp. 58–63.

[79] H. T. Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, Jan. 1982.

[80] F. Tramèr, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *CoRR*, vol. abs/1705.07204, 2017.

[81] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[82] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2015.

[83] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature learning approach for deep face recognition," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VII*, 2016, pp. 499–515.

[84] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[85] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[86] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE Symposium on Security and Privacy*, 2017.

[87] J. H. Ko, B. A. Mudassar, and S. Mukhopadhyay, "An energy-efficient wireless video sensor node for moving object surveillance," *IEEE Trans. Multi-Scale Computing Systems*, vol. 1, no. 1, pp. 7–18, 2015.

[88] J. H. Ko and S. Mukhopadhyay, "An energy-aware approach to noise-robust moving object detection for low-power wireless image sensor platforms," in *International Symposium on Low Power Electronics and Design, (ISLPED)*, 2016, pp. 194–199.

[89] T. Na, J. H. Ko, and S. Mukhopadhyay, "Cascade adversarial machine learning regularized with a unified embedding," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[90] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li, "Imagenet: A large-scale hierarchical image database," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[91] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 379–387.

[92] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," in *European Conference on Computer Vision (ECCV)*, vol. 8693, 2014, pp. 740–755.

[93] T. Haruta, T. Nakajima, J. Hashizume, T. Umebayashi, H. Takahashi, K. Taniguchi, M. Kuroda, H. Sumihiro, K. Enoki, T. Yamasaki, K. Ikezawa, A. Kitahara, M. Zen, M. Oyama, H. Koga, H. Tsugawa, T. Ogita, T. Nagano, S. Takano, and T. Nomoto,

"4.6 a 1/2.3inch 20mpixel 3-layer stacked cmos image sensor with dram," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 76–77.

[94] H. Lu, H. Zhang, S. Yang, and Z. Zheng, "Camera parameters auto-adjusting technique for robust robot vision," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 1518–1523.

[95] I. Shim, J. Y. Lee, and I. S. Kweon, "Auto-adjusting camera exposure for outdoor robotics using gradient information," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1011–1017.

[96] Y. Wu and J. K. Tsotsos, "Active control of camera parameters for object detection algorithms," *CoRR*, vol. abs/1705.05685, 2016.

[97] Z. Wang, Q. Hao, F. Zhang, Y. Hu, and J. Cao, "A variable resolution feedback improving the performances of object detection and recognition," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 232, no. 4, pp. 417–427, 2018.

[98] T. Na, J. H. Ko, and S. Mukhopadhyay, "Noise-robust and resolution-invariant image classification with pixel-level regularization," in *International Conference on Acoustics, Speech and Signal Processing, (ICASSP)*, 2018.

[99] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, no. 1, pp. 79–87, Mar. 1991.

[100] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural Comput.*, vol. 6, no. 2, pp. 181–214, Mar. 1994.

[101] A. Buades, B. Coll, and J. M. Morel, "A non-local algorithm for image denoising," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, 2005, 60–65 vol. 2.

[102] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Transactions on Image Processing*, vol. 16, no. 8, pp. 2080–2095, 2007.

[103] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proceedings of the Sixth International Conference on Computer Vision*, ser. ICCV '98, Washington, DC, USA: IEEE Computer Society, 1998, pp. 839–, ISBN: 81-7319-221-9.

[104] S. Milani, R. Bernardini, and R. Rinaldo, "Adaptive denoising filtering for object detection applications," in *2012 19th IEEE International Conference on Image Processing*, 2012, pp. 1013–1016.

[105] S Milyaev and I Laptev, "Towards reliable object detection in noisy images," *Pattern Recognition and Image Analysis*, vol. 27, pp. 713–722, Oct. 2017.

[106] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., Curran Associates, Inc., 2015, pp. 91–99.

[107] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 2818–2826.

[108] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *ICIP*, 2016.

[109] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, 2017.

[110] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *CoRR*, vol. abs/1607.08022, 2016.

[111] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI (3)*, ser. Lecture Notes in Computer Science, vol. 9351, Springer, 2015, pp. 234–241.

[112] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, ser. NIPS, 2014.

[113] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," in *CVPR*, IEEE Computer Society, 2017, pp. 3296–3297.

[114] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "MOT16: A benchmark for multi-object tracking," *arXiv:1603.00831 [cs]*, Mar. 2016, arXiv: 1603.00831.